

## In diesem Kapitel:

- SMTP - *emails versenden* (RFC 821)
- POP3 - *emails empfangen* (RFC 1725)
- *JavaMail*
  - *Was ist JavaMail*
  - *Senden von emails*
  - *Empfangen von emails*
  - *Passwort Authentifizierung*
  - *Adressen*
  - *Die URLName Klasse*
  - *Die Message Klasse*
  - *Das Part Interface*
  - *Multipart Messages und File Attachements*
  - *MIME Messages*
  - *Folders*

# 15. SMTP, POP3 und JavaMail

## 15.1. Um was geht's eigentlich?

Email war eine der Internet Killerapplikationen. Auch heute generiert email mehr Internet Verkehr als andere Internet Applikationen, ausser HTTP (Webseiten). Daher wurde schon sehr früh oft die Frage gestellt, ob man und wie man emails mit Java versenden kann.

Die Antwort ist sehr einfach: falls Sie die RFCs über SMTP (versenden von emails) und POP3 (empfangen von emails) gelesen haben und sich mit Sockets auskennen, sollten Sie die Antwort innerhalb einiger Stunden liefern können, auch ohne JavaMail.

Aber falls Sie komplexere email Anwendungen schreiben möchten, bietet JavaMail sehr viel fix fertig, was Sie sonst mühsam selbst implementieren müssten. Und wer liest schon RFC's? Sie könnten ja auch HTTP leicht mit Sockets implementieren!

JavaMail ist eine Standarderweiterung von Java (ursprünglich 1.1) und gestattet es secht komplexe email Anwendungen zu entwickeln. Heute gehört JavaMail zur J2EE, zur Java Enterprise Edition. Die Referenzimplementation von Sun wird mit Sockets und Streams implementiert. Anwendungen können JavaMail verwenden, um mit SMTP und IMAP Servern zu kommunizieren, emails zu senden und zu empfangen. Der Einsatz von JavaMail reduziert den Aufwand, den man hätte, falls man sich auf Low Level Programmierung, mit Sockets und Streams, einlassen würde. Das System lässt sich leicht erweitern in Richtung proprietärer email Systeme, wie etwa Lotus Notes, cc:Mail und andere. Auch NNTP (News) kann leicht implementiert werden.

Es ist also sehr leicht, ein email System analog zu Eudora oder Outlook Express zu erstellen. Aber man kann auch effiziente email Systeme wie Majordomo, List Server und ähnliches bauen. JavaMail können Sie aber auch in andere Applikationen einbauen, Applikationen, welche einen mail Anschluss benötigen, beispielsweise um Benutzer über Änderungen zu informieren. Als Applikation kann dabei auch ein Applet gemeint sein, unter Ausnutzung des

# NETZWERKPROGRAMMIERUNG IN JAVA

SMTP Servers des CODEBASE Servers. Auch NewsReader lassen sich mit JavaMail implementieren.

Auch die Low Level Implementierung ist denkbar einfach, sofern wir uns auf die wesentlichen Funktionen beschränken.

## 15.2. SMTP (Simple Message Transfer Protocol) gemäss RFC 821

Schauen wir uns als erstes den RFC 821, die Beschreibung des Simple Message Protocols an. Sie finden alle wichtigen RFCs auf dem Web, beispielsweise bei Switch. Der Einfachheit halber finden Sie diesen RFC auch als PDF bei den Unterlagen.

Der RFC stammt aus dem Jahre 1982 und wurde von Jonathan Postel vorgeschlagen.

1. INTRODUCTION.....	1
2. THE SMTP MODEL .....	2
3. THE SMTP PROCEDURE .....	4
3.1. Mail .....	4
3.2. Forwarding .....	7
3.3. Verifying and Expanding .....	8
3.4. Sending and Mailing .....	11
3.5. Opening and Closing .....	13
3.6. Relaying .....	14
3.7. Domains .....	17
3.8. Changing Roles .....	18
4. THE SMTP SPECIFICATIONS .....	19
4.1. SMTP Commands .....	19
4.1.1. Command Semantics .....	19
4.1.2. Command Syntax .....	27
4.2. SMTP Replies .....	34
4.2.1. Reply Codes by Function Group .....	35
4.2.2. Reply Codes in Numeric Order .....	36
4.3. Sequencing of Commands and Replies .....	37
4.4. State Diagrams .....	39
4.5. Details .....	41
4.5.1. Minimum Implementation .....	41
4.5.2. Transparency .....	41
4.5.3. Sizes .....	42
APPENDIX A: TCP .....	44
APPENDIX B: NCP .....	45
APPENDIX C: NITS .....	46
APPENDIX D: X.25 .....	47
APPENDIX E: Theory of Reply Codes .....	48
APPENDIX F: Scenarios .....	51
GLOSSARY .....	64
REFERENCES .....	67

Schauen wir uns den SMTP Ablauf an, so wie er in Kapitel 3 beschrieben wird.

Der Ablauf besteht aus mehreren (drei) Schritten:

1. als erstes wird die Transaktion mit dem `mail` Befehl gestartet.
2. dann folgt ein oder mehrere `RCPT` Befehle, mit denen die Empfänger spezifiziert werden.
3. schliesslich folgen die Daten nach dem `DATA` Befehl.

Schauen wir uns ein konkretes Beispiel an, direkt aus dem RFC

### Beispiel 1 SMTP

In diesem Beispiel wird ein email an Smith am Host Alpha.ARPA, Jones, Green und Brown am Host Beta.ARPA gesendet. (S=Sender; R=Receiver)

# NETZWERKPROGRAMMIERUNG IN JAVA

```
S: MAIL FROM:<Smith@Alpha.ARPA>
R: 250 OK
S: RCPT TO:<Jones@Beta.ARPA>
R: 250 OK
S: RCPT TO:<Green@Beta.ARPA>
R: 550 No such user here
S: RCPT TO:<Brown@Beta.ARPA>
R: 250 OK
S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: <CRLF>.<CRLF>
R: 250 OK
```

**Selbsttestaufgabe 1** Schreiben Sie sich ein email mit Hilfe von Telnet. Gehen Sie gemäss obigem Schema vor. (`smtp 25/tcp mail`: siehe Anhang im RFC)

**Selbsttestaufgabe 2** Welche Befehle werden im RFC definiert. Testen Sie mindestens drei der Befehle.

Eine Liste der Fehler- und Status-Codes finden Sie im RFC.

Falls Ihr Telnet email System funktioniert, können Sie das selbe mit Hilfe von Sockets realisieren (siehe Aufgaben).

Soweit zu SMTP und RFC 821

## 15.3. POP3 (Post Office Protocol V3) gemäss RFC 1725

J. Myers (CMU) und M. Rose haben 1994 die Version 3 des POP Protokolls beschrieben. Auch hier lässt sich der RFC leicht verstehen, wenn man sich auf die wesentlichen Teile konzentriert und beispielsweise mit einem einfachen Beispiel anfängt.

### Beispiel 2 Beispiel für eine POP3 Session

```
S: <wait for connection on TCP port 110>
C: <open connection>
//Eingabe von Benutzername und Passwort
S: +OK POP3 server ready <1896.697170952@dbc.mtview.ca.us>
C: APOP mrose c4c9334bac560ecc979e58001b3e22fb
S: +OK mrose's maildrop has 2 messages (320 octets)
C: STAT
S: +OK 2 320
C: LIST
S: +OK 2 messages (320 octets)
S: 1 120
S: 2 200
S: .
C: RETR 1
S: +OK 120 octets
S: <the POP3 server sends message 1>
S: .
C: DELE 1
S: +OK message 1 deleted
C: RETR 2
S: +OK 200 octets
S: <the POP3 server sends message 2>
S: .
```

# NETZWERKPROGRAMMIERUNG IN JAVA

```
C: DELE 2
S: +OK message 2 deleted
C: QUIT
S: +OK dewey POP3 server signing off (maildrop empty)
C: <close connection>
S: <wait for next connection>
```

**Selbsttestaufgabe 3** Fragen Sie Ihre email Konto mit Hilfe von Telnet ab. Gehen Sie gemäss obigem Schema vor. ( `pop3 110/tcp postoffice` )

**Selbsttestaufgabe 4** Welche Befehle werden im RFC definiert. Testen Sie mindestens drei der Befehle.

Eine Liste der Fehler- und Status-Codes finden Sie im RFC.

Falls Ihr Telnet email System funktioniert, können Sie das selbe mit Hilfe von Sockets realisieren (siehe Aufgaben).

Soweit zu POP3 und RFC 1725

## 15.4. Was ist das JavaMail API?

Das JavaMail API ist eine Darstellung der Basiskomponenten eines email Systems auf einer sehr hohen Ebene. Die Komponenten werden durch abstrakte Klassen dargestellt. Diese stehen im `javax.mail` Paket.

### Wie gelangt man am schnellsten zu einer Übersicht über ein fremdes Programmpaket?

Falls das Paket mit JavaDoc beschrieben wird, ist die JavaDoc die beste Dokumentation. Eventuell müssen oder können Sie das bestehende Paket nachdokumentieren oder einfach JavaDoc starten. Allerdings könnten Sie auch mit Reverse Engineering ein Klassendiagramm erzeugen, aus dem dann die Struktur des Paketes ersichtlich wird.

Für das reverse Engineering können Sie eines der gängigen UML Werkzeuge verwenden, beispielsweise Rational Rose oder TogetherJ. Die resultierenden Darstellungen dieser beiden Werkzeuge sind etwas unterschiedlich, wegen der nicht so ganz Standard konformen Darstellung im TogetherJ.

Die Modelle sind sehr umfangreich, typische für reale Anwendungen. Aber das Klassendiagramm, eventuell in einer ersten Version ohne Methoden und Datenfelder, zeigt bereits eine gute Übersicht über die vorhandenen Klassen und Beziehungen untereinander.

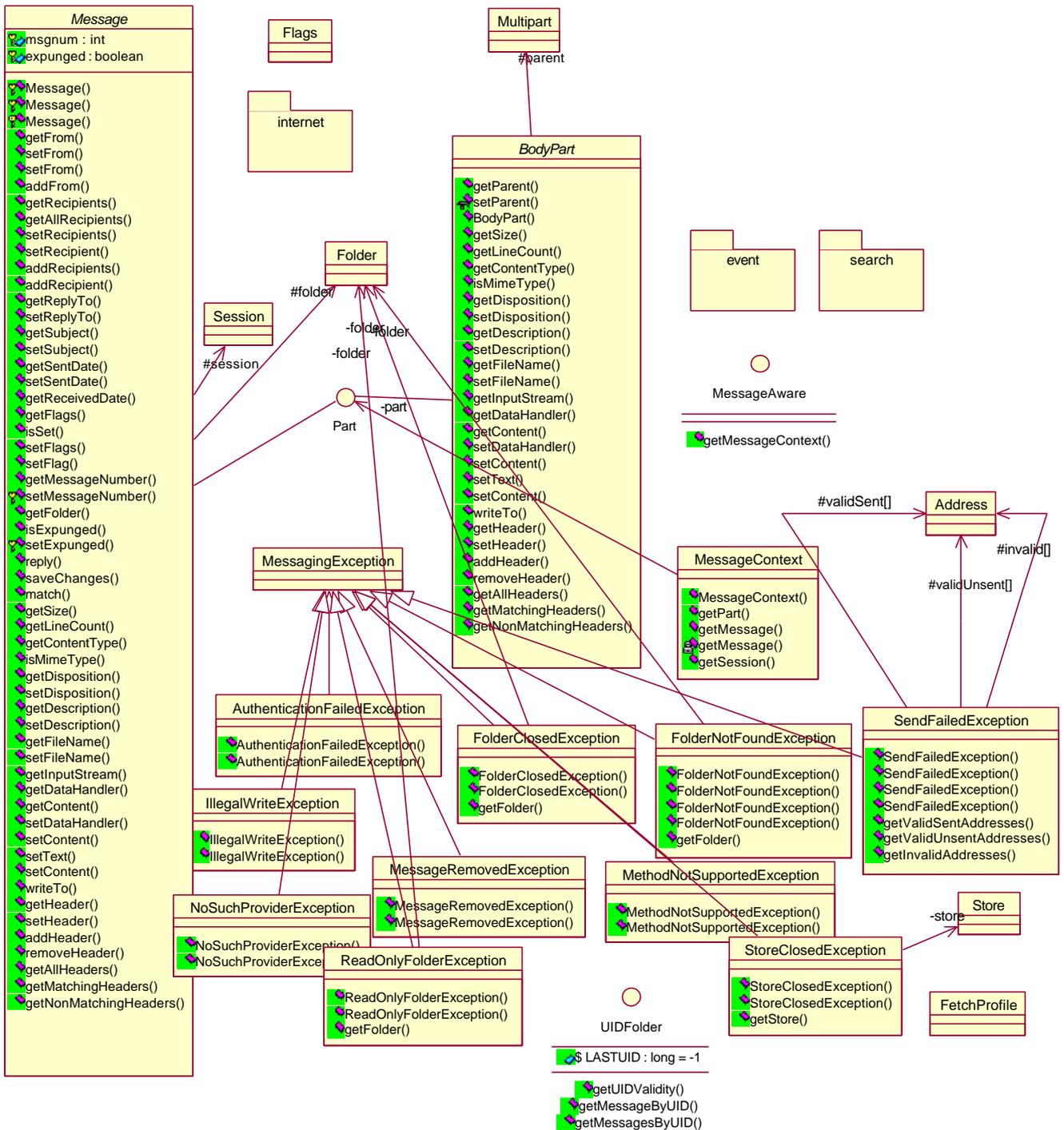
Sie finden beide Modelle bei den Unterlagen. Die aktuelle Version von Rational Rose und eine Trial Version von ToGetherJ finden Sie ebenfalls bei den Unterlagen.

Wir werden in den folgenden Abschnitten dieses Paket genauer anschauen und besprechen, in der Regel anhand von Beispielen.

Damit Sie die Beispiele testen können, sollten Sie eine Socket Verbindung zu Ihrem mail Server haben. Dazu sollten Sie vorgängig die Aufgaben mit der SMTP und POP3 Verbindung mit Sockets gemacht haben. Diese beiden Aufgaben testen Ihr Umfeld und garantieren das Funktionieren der folgenden Beispiele.

# NETZWERKPROGRAMMIERUNG IN JAVA

Abbildung 1 Übersicht über das Paket mail (Auszug aus javax Anteil)



# NETZWERKPROGRAMMIERUNG IN JAVA

Message
msgnum : int
expunged : boolean
Message()
Message()
Message()
getFrom()
setFrom()
setFrom()
addFrom()
getRecipients()
getAllRecipients()
setRecipients()
setRecipient()
addRecipients()
addRecipient()
getReplyTo()
setReplyTo()
getSubject()
setSubject()
getSentDate()
setSentDate()
getReceivedDate()
getFlags()
isSet()
setFlags()
setFlag()
getMessageNumber()
setMessageNumber()
getFolder()
isExpunged()
setExpunged()
reply()
saveChanges()
match()
getSize()
getLineCount()
getContentType()
isMimeType()
getDisposition()
setDisposition()
getDescription()
setDescription()
getFileName()
setFileName()
getInputStream()
getDataHandler()
getContent()
setDataHandler()
setContent()
setText()
setContent()
writeTo()
getHeader()
setHeader()
addHeader()
removeHeader()
getAllHeaders()
getMatchingHeaders()
getNonMatchingHeaders()

Viele Klassen sind abstrakt. Beispielsweise repräsentiert die abstrakte Klasse `javax.mail.Message` (Sie erkennen die Abstraktheit der Klasse an der kursiven Schreibweise) eine generelle email Message. Die (abstrakte) Klasse definiert mehrere `set...` und mehrere `get...` Methoden, mit deren Hilfe "Umschlagsinformationen" (z.B. `set Recipient`, `getRecipient`, `setFrom`, `getFrom`) gesetzt oder abgefragt werden können.

Die abstrakte Klasse `javax.mail.Folder` repräsentiert einen Message Container. Ziel eines solchen Folders ist es, Messages zu sammeln und gegebenenfalls zu verschieben oder zu löschen.

Diese Klassen sind alle abstrakt, weil damit freigestellt wird, wie die Implementierungen auf einer bestimmten Hardware und Systemsoftware Plattform auszusehen hat. Die Spezifikation bleibt somit allgemein, abstrakt. Die konkrete Implementation muss die Details festlegen.

Die Protokolle werden mit sogenannten Service Providern implementiert. Die Spezifikation der Server Provider steht allgemein zur Verfügung. Ein Service Provider besteht aus einer Gruppe konkreter Unterklassen der abstrakten Klassen. des JavaMail APIs. Beispiele sind: SMTP und IMAP. Diese finden Sie als Referenzimplementationen im Paket `com.sun.mail`. Andere Protokolle werden beispielsweise von andern Anbietern offeriert. Andere wie beispielsweise POP sind sowohl von Sun als auch von Dritten erhältlich. Die Idee der abstrakten Definition des JavaMail APIs und auch die Stärke dieser Spezifikation besteht darin, dass Sie sich nicht um die Details der Implementierung kümmern müssen. Sie können sich an die allgemeinen Klassen halten. Sie können sogar den Provider wechseln, ohne dass Ihre Applikation neu übersetzt werden muss, sagen wir mal, im Idealfall.

Damit JavaMail Meldungen asynchron verarbeiten kann (sie kommen asynchron aus dem Internet), benötigt das JavaMail API einen ereignisgesteuerten Callback Mechanismus. Das Schema entspricht jenem von AWT oder Java Beans. Das Paket `javax.mail.event` definiert ungefähr ein Duzend unterschiedliche Mailereignisse, Listenerinterfaces und Adapterklassen.

Damit JavaMail auch komplexere emails (neben Text auch Video, Bilder, ...), verwendet JavaMail das Java Beans Activation Framework. (JAF). Sie müssen dieses installieren, damit Sie JavaMail erfolgreich testen können.

Die Klassen des Pakets `javax.mail` finden Sie im Archiv `mail.jar`.

Sie müssen dieses Archiv in Ihren Klassenpfad aufnehmen oder besser bei der Ausführung Ihres Programms im Klassenpfad

# NETZWERKPROGRAMMIERUNG IN JAVA

angeben (ab Java2 arbeitet man zunehmend nicht mehr mit dem CLASSPATH). Sie können auch einfach das Archiv in Ihr Laufzeit Java Verzeichnis verschieben, ins Verzeichnis `java/lib/ext`. Alternativ dazu kopieren Sie Archive in das `java/lib` Verzeichnis von JDK.

Das Activation Framework finden Sie entweder bei den Unterlagen oder auf dem Java Web Site.

Falls Sie weitere Provider suchen, hier ist eine kurze Liste:

**Tabelle 1** JavaMail Provider

Produkt	URL	Protokolle	Lizenzkosten
JavaMail (Sun)	<a href="http://java.sun.com/products/javamail/">http://java.sun.com/products/javamail/</a>	SMTP, IMAP	gratis
POP3 (Sun)	<a href="http://java.sun.com/products/javamail/">http://java.sun.com/products/javamail/</a>	POP3	gratis
Jason Diamond POP3 Provider für JavaMail	<a href="http://injektilo.org/pop3mail.html">http://injektilo.org/pop3mail.html</a>	POP3	gratis
ICE Engineering Inc. MH JavaMail Provider	<a href="http://www.trustice.com/java/icemh">http://www.trustice.com/java/icemh</a>	MH	public domain
Linkable Software POP3 Provider	<a href="http://www.linkable.com/products/pop3/">http://www.linkable.com/products/pop3/</a>	POP3, mbox	Evaluation
dog.mail	<a href="http://www.dog.net.uk/knife">http://www.dog.net.uk/knife</a>	POP3, NNTP, mbox	Mozilla Public Licence

## 15.5. Senden von emails

Die Grundfunktion des emails besteht im Versenden einer Nachricht. Jeder Menge Programme sind in der Lage Nachrichten zu versenden; empfangen werden sie in der Regel von Programmen wie Eudora, OutlookExpress oder Netscape Messenger. Tools wie Satan, ein Sicherheitstool, laufen im Hintergrund und informieren den Manager, falls eine Ausnahme eintritt. Viele Web Sites informieren Sie, falls signifikante Änderungen durchgeführt wurden.

Das JavaMail API stellt alles zur Verfügung, was ein Anwendungsprogramm benötigt, um Meldungen zu versenden. Dies geschieht in folgenden acht Schritten:

1. setzen der `mail.host` Property, um auf den gewünschten Mail Server zu verweisen.
2. starten einer email Session mit der `Session.getInstance()` Methode
3. kreieren eines neuen Message Objekts, durch Instanzierung einer ihrer konkreten Unterklassen.
4. setzen der From Adresse
5. setzen der To Adresse
6. setzen des Betreff Feldes : Subject
7. setzen des Inhalts der Meldung / Nachricht
8. senden der Nachricht mit der `Transport.send()` Methode.

Die Reihenfolge der Schritte ist eher locker, bis auf den letzten. Das dürfte aus dem Kontext klar sein. Jeder dieser Schritte ist auch sehr einfach.

# NETZWERKPROGRAMMIERUNG IN JAVA

Das Setzen der Property ist typisch für Java2. Alles was Sie zu setzen haben ist der Mailhost. Über diesen werden Sie Ihre Nachrichten versenden. Diese Eigenschaft ist ein `java.util.Properties` Objekt, keine Umgebungsvariable.

## Beispiel 3 Setzen des Mailhost im HTA Netzwerk

```
Properties props = new Properties();
props.put("mail.host", "mailhost.hta.fhz.ch");
```

Diese Eigenschaft wird beim Erzeugen des Session Objekts benötigt:

```
Session mailConnection = Session.getInstance(props, null);
```

Das `Session` Objekt stellt eine Kommunikation zwischen einem Programm und einem Mailserver dar. Das zweite Argument der `getInstance()` Methode wird hier auf `null` gesetzt. Es entspricht dem `javax.mail.Authenticator`, mit dem das Passwort für den Zugriff auf den Mailserver erfragt wird. Diesen Fall werden wir weiter hinten noch genauer diskutieren. In der Regel benötigt man zum Senden von Nachrichten weder Benutzernamen noch Passwort.

Mit Hilfe des `Session` Objekts wird ein neues `Message` Objekt konstruiert:

```
Message msg = new MimeMessage(mailConnection); // von oben
```

Wir verwenden `MimeMessage`, weil wir komplexe Internet emails versenden wollen. Hier muss man Aussagen zum Format machen.

Nachdem wir nun ein `Message` Objekt haben, müssen wir die Felder und den Inhalt der Nachricht definieren. Die `From` und die `To` Adresse sind `javax.mail.internet.InternetAddress` Objekte. Als Adresse können Sie entweder lediglich eine Internet / email Adresse, oder aber einen Namen plus eine Internet / email Adresse angeben:

```
Address bill = new InternetAddress("god@microsoft.com", "Bill Gates");
Address jo = new InternetAddress("joller@joller-voss.ch", "Josef Joller");
```

Mit der `setFrom()` Methode kann der Absender gesetzt werden. Dabei ist es Ihnen freigestellt, einen Absender zu erfinden, da der Absender nicht überprüft wird:

```
msg.setFrom(jo);
```

Das Empfängerfeld ist etwas komplexer. Sie müssen neben der Adresse auch noch festlegen, ob es sich um ein `cc:`, `bcc:` oder was auch immer handelt. Der Typus des Empfängers wird mit Hilfe einer Konstante der `Message.RecipientType` Klasse festgelegt:

```
Message.RecipientType.TO
Message.RecipientType.CC
Message.RecipientType.BCC
```

## Beispiel 4 Das Empfängerfeld

```
msg.setRecipient(Message.RecipientType.TO, jo);
```

Das "Betreff" / Subject Feld wird einfach als Text, als Zeichenkette gesetzt:

```
msg.setSubject("Bitte um eine Antwort!");
```

# NETZWERKPROGRAMMIERUNG IN JAVA

Schliesslich wird der Inhalt der Nachricht ebenfalls als Zeichenkette gesetzt, wobei allerdings noch der MIME Type gesetzt werden muss.

## Beispiel 5 Eine einfache Meldung

```
msg.setContent("ueberweisen Sie mir bitte ein anstaendiges Gehalt",  
              "text/plain");
```

Nun sind wir soweit: wir können die Nachricht versenden. Die statische Methode `Transport.send()` verbindet das Programm mit dem in dem `mail.property` genannten Host und sendet die Nachricht.

Das folgende Beispiel, eine selbständige Applikation, fügt alle Teile, die wir oben besprochen haben, zusammen und sendet die einfache Textmeldung.

Beachten Sie, dass die Archive von JavaMail und Java Beans Activation im angemeldet werden müssen (CLASSPATH oder wie auch immer).

Ich habe die Bibliotheken als Archive in das Verzeichnis `JBuilder4/JDK1.3/lib` kopiert und anschliessen bei den Projekteigenschaften hinzugefügt.

Unter diesen Umständen kompiliert das Programm fehlerfrei:

## Beispiel 6 Einfache Text Meldung

```
package einfachestextmail;  
  
/**  
 * Title:          Einfaches Text Mail  
 * Description:    Beispiel für den Einsatz des JavaMail APIs.  
 * Aufbau:  
 * 0) setzen des Mail Hosts (ausgehende Meldungen)  
 * 1) definieren und kreieren der Internet Adressen der Sender und  
Empfänger  
 * 2) Setzen der From, To und Empfängertypus Felder (TO)  
 * 3) setzen des Nachrichtentextes  
 * 4) versenden der Nachricht  
 * Copyright:     Copyright (c) 2000  
 * Company:       Joller-Voss GmbH  
 * @author J.M.Joller  
 * @version 1.0  
 */  
  
import javax.mail.*;  
import javax.mail.internet.*;  
import java.util.*;  
  
public class EinfachesTextMail {  
  
    public static void main(String[] args) {  
  
        Properties props = new Properties();  
        props.put("mail.host", "mailhost.hta.fhz.ch");  
  
        try {  
  
            // 0)
```

# NETZWERKPROGRAMMIERUNG IN JAVA

```
Session mailConnection = Session.getInstance(props, null);
// 1)
Address bill = new InternetAddress("god@microsoft.com", "Bill
Gates");
Address elliotte = new InternetAddress("joller@joller-voss.ch");

Message msg = new MimeMessage(mailConnection);
// 2)
msg.setFrom(bill);
msg.setRecipient(Message.RecipientType.TO, elliotte);
msg.setSubject("Was soll das?.");
// 3)
msg.setContent("Warum bezahlt die MIGROS so schlecht?",
"text/plain");
// 4)
Transport.send(msg);

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

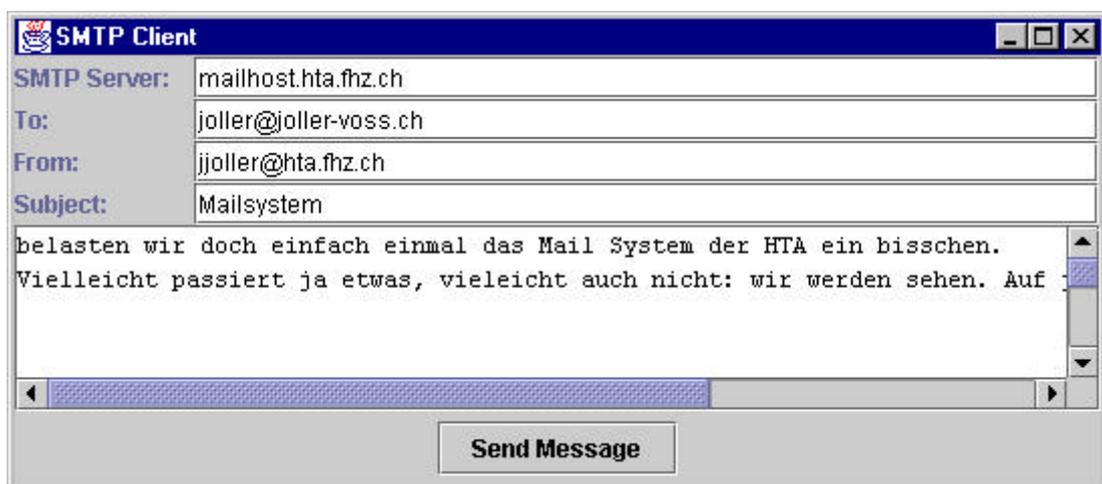
Das Ergebnis ist eine Standardmeldung gemäss RFC 821/822

```
Date: Sun, 10 Dec 2000 15:45:21 +0100
From: Bill Gates god@microsoft.com
To: joller@joller-voss.ch
Subject: Was soll das?
```

## 15.5.1. Senden einer Meldung aus einer Applikation

Das Beispiel 6 oben ist eine einfache Anwendung, welche eine feste Nachricht von einer festen Adresse an eine feste Adresse sendet mit einem fixen Betrifft: Feld. Aber die Änderung des Beispiels, um dynamische Meldungen an variable Adressen zu versenden ist recht einfach. Wir könnten beispielsweise die Datenfelder aus der Kommandozeile oder über ein GUI einlesen. Als erstes ergänzen wir das Programm durch ein GUI.

### Abbildung 2 Einfache GUI für den SMTP Client



# NETZWERKPROGRAMMIERUNG IN JAVA

In diesem Beispiel kapseln wir alle JavaMail API Anteile in die Klasse `SendAction` speziell in deren Methode `actionPerformed()`.

## Beispiel 7 Ein grafischer SMTP Client

```
package smtpclient;

/**
 * Title:          SMTP Client
 * Description:    Ein einfaches GUI zum SMTP Programm
 * Copyright:     Copyright (c) 2000
 * Company:      Joller-Voss GmbH
 * @author J.M.Joller
 * @version 1.0
 * Title:          Einfaches Text Mail
 * Description:    Beispiel für den Einsatz des JavaMail APIs.
 * Aufbau:
 * 0) setzen des Mail Hosts (ausgehende Meldungen)
 * 1) definieren und kreieren der Internet Adressen der Sender und
Empfänger
 * 2) Setzen der From, To und Empfängertypus Felder (TO)
 * 3) setzen des Nachrichtentextes
 * 4) versenden der Nachricht
 */

import javax.mail.*;
import javax.mail.internet.*;
import java.util.*;
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;

public class SMTPClient extends JFrame {

    private JButton    sendButton    = new JButton("Send Message");
    private JLabel     fromLabel     = new JLabel("From: ");
    private JLabel     toLabel       = new JLabel("To: ");
    private JLabel     hostLabel     = new JLabel("SMTP Server: ");
    private JLabel     subjectLabel  = new JLabel("Subject: ");
    private JTextField fromField     = new JTextField(40);
    private JTextField toField       = new JTextField(40);
    private JTextField hostField     = new JTextField(40);
    private JTextField subjectField  = new JTextField(40);
    private JTextArea  message      = new JTextArea(40, 72);
    private JScrollPane jsp         = new JScrollPane(message);

    public SMTPClient() {

        super("SMTP Client");
        Container contentPane = this.getContentPane();
        contentPane.setLayout(new BorderLayout());

        JPanel labels = new JPanel();
        labels.setLayout(new GridLayout(4, 1));
        labels.add(hostLabel);

        JPanel fields = new JPanel();
```

# NETZWERKPROGRAMMIERUNG IN JAVA

```
fields.setLayout(new GridLayout(4, 1));
String host = System.getProperty("mail.host", "");
hostField.setText(host);
fields.add(hostField);

labels.add(toLabel);
fields.add(toField);

String from = System.getProperty("mail.from", "");
fromField.setText(from);
labels.add(fromLabel);
fields.add(fromField);

labels.add(subjectLabel);
fields.add(subjectField);

Box north = Box.createHorizontalBox();
north.add(labels);
north.add(fields);

contentPane.add(north, BorderLayout.NORTH);

message.setFont(new Font("Monospaced", Font.PLAIN, 12));
contentPane.add(jsp, BorderLayout.CENTER);

JPanel south = new JPanel();
south.setLayout(new FlowLayout(FlowLayout.CENTER));
south.add(sendButton);
sendButton.addActionListener(new SendAction());
contentPane.add(south, BorderLayout.SOUTH);

this.pack();
}

class SendAction implements ActionListener {

    public void actionPerformed(ActionEvent evt) {

        try {
            Properties props = new Properties();
            props.put("mail.host", hostField.getText());

            Session mailConnection = Session.getInstance(props, null);
            final Message msg = new MimeMessage(mailConnection);

            Address to = new InternetAddress(toField.getText());
            Address from = new InternetAddress(fromField.getText());

            msg.setContent(message.getText(), "text/plain");
            msg.setFrom(from);
            msg.setRecipient(Message.RecipientType.TO, to);
            msg.setSubject(subjectField.getText());

            // This can take a non-trivial amount of time so
            // spawn a thread to handle it.
            Runnable r = new Runnable() {
                public void run() {
                    try {
                        Transport.send(msg);
                    }
                    catch (Exception e) {

```

# NETZWERKPROGRAMMIERUNG IN JAVA

```
        e.printStackTrace();
    }
}
};
Thread t = new Thread(r);
t.start();

message.setText("");
}
catch (Exception e) {
    // in einer produktiven Umgebung müsste dies verbessert werden.
    e.printStackTrace();
}
}
}

public static void main(String[] args) {

    SMTPClient client = new SMTPClient();
    // Java 1.3.
    client.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    client.show();

}
}
```

Dieses Programm ist alles andere als ein ideales Programm:

- das GUI ist zuwenig klar vom eigentlichen SMTP getrennt
- Fehlerdialoge sind zu rudimentär implementiert

Aber diese Erweiterungen würden nichts zum Verständnis der Funktionsweise von JavaMail API beitragen. Falls Sie dies stört, können Sie das Programm ja ergänzen.

## 15.5.2. Senden von emails aus einem Applet

Vom Standpunkt eines GUIs betrachtet, bestehen kaum Unterschiede zwischen einer Anwendung und einem Applet. Allerdings hat der Security Manager des Browsers in der Regel etwas gegen zu viel freien Zugriff. Ein Applet kann also lediglich mit dem Host kommunizieren, von dem es stammt.

Zum Glück verfügen die meisten Server, welche Web Dienste anbieten auch über Mail Dienste. Falls dies zutrifft, dann ist die Realisierung eines email Systems als Applet einfach. Allerdings müssen wir einige Archive vom Server zum Client herunterladen, da beispielsweise das Activation Framework in den Browsern nicht vorhanden ist.

### Beispiel 8 HTML für das Mail Applet

```
<APPLET CODEBASE="."
        CODE=SMTPApplet
        ARCHIVE="activation.jar,mail.jar"
        WIDTH=600
        HEIGHT=400
>

    <PARAM NAME="to"           VALUE="joller@joller-voss.ch">
    <PARAM NAME="subject"      VALUE="Was soll das?">
```

# NETZWERKPROGRAMMIERUNG IN JAVA

```
<PARAM NAME="from" VALUE=jjoller@hta.fhz.ch>
</APPLET>
```

## Bemerkung 1 Netscape

Verschiedene Browser haben verschiedene Sicherheitsmechanismen implementiert. Netscape hat sich dabei in der Version 4.x nicht an die Grundsätze des Java Security Modells gehalten. Deswegen kann ein email Applet im Falle des Netscape Browsers nicht funktionieren.

Wie Sie oben sehen können, werden die Parameternamen und Parameterwerte im Applet Tag angegeben.

## Abbildung 3 SMTPApplet



## Beispiel 9 SMTPApplet

```
package smtpapplet;
```

```
/**
```

```
 * Title: SMTP Applet
```

```
 * Description: Das Applet liest die Werte der Parameter
```

```
 * From:
```

```
 * To:
```

```
 * und Subject:
```

```
 * Diese spezifizieren die entsprechenden Werte für das SMTPApplet
```

```
 * Copyright: Copyright (c) 2000
```

```
 * Company: Joller-Voss GmbH
```

```
 * @author J.M.Joller
```

```
 * @version 1.0
```

```
*/
```

```
import java.applet.*;
```

```
import javax.mail.*;
```

```
import javax.mail.internet.*;
```

```
import java.util.Properties;
```

# NETZWERKPROGRAMMIERUNG IN JAVA

```
import java.awt.event.*;
import java.awt.*;

public class SMTPApplet extends Applet {

    private Button    sendButton    = new Button("Send Message");
    private Label    fromLabel    = new Label("From: ");
    private Label    subjectLabel = new Label("Subject: ");
    private TextField fromField    = new TextField(40);
    private TextField subjectField = new TextField(40);
    private TextArea message      = new TextArea(30, 60);

    private String toAddress = "";

    public SMTPApplet() {

        this.setLayout(new BorderLayout());

        Panel north = new Panel();
        north.setLayout(new GridLayout(3, 1));

        Panel n1 = new Panel();
        n1.add(fromLabel);
        n1.add(fromField);
        north.add(n1);

        Panel n2 = new Panel();
        n2.add(subjectLabel);
        n2.add(subjectField);
        north.add(n2);

        this.add(north, BorderLayout.NORTH);

        message.setFont(new Font("Monospaced", Font.PLAIN, 12));
        this.add(message, BorderLayout.CENTER);

        Panel south = new Panel();
        south.setLayout(new FlowLayout(FlowLayout.CENTER));
        south.add(sendButton);
        sendButton.addActionListener(new SendAction());
        this.add(south, BorderLayout.SOUTH);

    }

    public void init() {

        String subject = this.getParameter("subject");
        if (subject == null) subject = "";
        subjectField.setText(subject);
    }
}
```

# NETZWERKPROGRAMMIERUNG IN JAVA

```
toAddress = this.getParameter("to");
if (toAddress == null) toAddress = "";

String fromAddress = this.getParameter("from");
if (fromAddress == null) fromAddress = "";
fromField.setText(fromAddress);

}

class SendAction implements ActionListener {

    public void actionPerformed(ActionEvent evt) {

        try {
            Properties props = new Properties();
            props.put("mail.host", getCodeBase().getHost());

            Session mailConnection = Session.getInstance(props, null);
            final Message msg = new MimeMessage(mailConnection);

            Address to = new InternetAddress(toAddress);
            Address from = new InternetAddress(fromField.getText());

            msg.setContent(message.getText(), "text/plain");
            msg.setFrom(from);
            msg.setRecipient(Message.RecipientType.TO, to);
            msg.setSubject(subjectField.getText());

            // Starten des Threads
            // Das kann etwas dauern.
            Runnable r = new Runnable() {
                public void run() {
                    try {
                        Transport.send(msg);
                    }
                    catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            };
            Thread t = new Thread(r);
            t.start();

            message.setText("");
        }
        catch (Exception e) {
            // zu einfach
            e.printStackTrace();
        }
    }
}
```

}

Da die Applikation keine Swing Komponenten verwendet, müssen weniger Archive herunter geladen werden. Die Abbildung oben ist natürlich nicht echt, weil Sie einfach lokal generiert wurde.

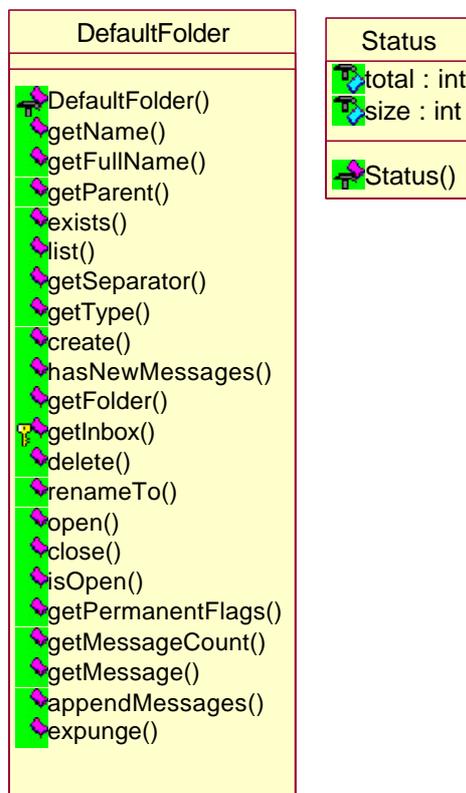
Das Applet kann nicht funktionieren, falls die Archive nicht herunter geladen werden können. Diese müssen also lokal vorhanden sein.

Zudem muss der Webserver auch noch ein Mailserver sein.

## 15.6. Empfangen von emails

Das Empfangen von emails ist oder kann komplexer sein als das Senden. Zum Senden benötigt man fast nur einen Server. Mit dem HELLO Befehl kann man fast auf jeden Mailserver zugreifen. Auf einen POP Server kann man nur zugreifen, falls man einen Benutzernamen und ein Passwort besitzt. Ein SMTP Client kann 15 verschiedene Befehle implementieren, wobei lediglich 5 benötigt werden. Beim POP3 hat man die Auswahl aus 12 Befehlen, von denen man aber die meisten benötigt. Im Falle von IMAP sieht die Situation noch komplexer aus: IMAP kennt 24 unterschiedliche Befehle.

### Abbildung 4 POP3 Provider von Sun



Das JavaMail API geht davon aus, dass der Server (IMAP, NNTP, ...) sowohl Header Informationen als auch die eigentlichen Meldungen verwalten kann. Der Client kann also entweder die Header Information oder die Meldung selbst abfragen. Auch Suchen auf dem Server sollte möglich sein. Der Server speichert die Meldungen, nicht der Client. Das System liefert weniger Hilfsklassen, um beispielsweise POP3 Meldungen auf der Clientseite zu verwalten. Aber das lässt sich natürlich selber, unabhängig vom JavaMail API implementieren.

Da viele ISPs nicht bereit sind Plattenspeicher für IMAP Clients zur Verfügung zu stellen, sind die meisten Mailsysteme heute immer noch POP3 basiert. Aus JavaMail Sicht ist ein IMAP ein POP3 plus Verwaltungsfunktionen. Sun hatte ursprünglich keinen POP3 Provider zur Verfügung gestellt. Falls Sie lange genug suchen, finden Sie jedoch auf dem Web mehrere (siehe auch die Tabelle weiter vorne). Der Unterschied besteht in der Funktionalität: der POP3 Provider von Sun verfügt nur über eine minimale Funktionalität.

Das Lesen einer (einzelnen) Mailbox geschieht grob in 12 Schritten:

1. Properties definieren
2. Authenticator konstruieren, um die Verbindung aufbauen zu können

# NETZWERKPROGRAMMIERUNG IN JAVA

3. Session Objekt bestimmen: `Session.getDefaultInstance()`
4. mit dem Session Objekt den Speicher / Store bestimmen: `getStore()`
5. verbinden mit dem Nachrichtenspeicher (`connect` mit dem `store`)
6. bestimmen der INBOX, des INBOX Folders, mit der `getFolder()` Methode
7. öffnen des INBOX Folders
8. öffnen des gewünschten Folders innerhalb des INBOX Folders. Da die Folder verschachtelt sein können, muss man diesen Schritt eventuell mehrfach ausführen.
9. holen der Meldungen vom Folder als ein Array von Message Objekten.
10. bearbeiten der Meldungen im Array
11. schliessen des Folders
12. schliessen des Stores

Jeder dieser Schritte für sich genommen ist recht einfach.

Im ersten Schritt könnte man beispielsweise folgende Eigenschaften setzen:

1. `mail.host`
2. `mail.store.protocol`
3. `mail.user`
4. `mail.pop3.user`
5. `mail.pop3.host`

Properties
Properties()
Properties()
getProperty()
getProperty()
list()
list()
load()
propertyNames()
save()
setProperty()
store()

Sie können aber genauso ein leeres `Properties` Objekt kreieren und die Angaben später vervollständigen.

## Beispiel 10 Definition des (leeren) Properties Objekts

```
Properties props = new Properties();
```

java.lang.Object
<b>javax.mail.Authenticator</b>
-requestingSite: java.net.InetAddress
-requestingPort: int
-requestingProtocol: java.lang.String
-requestingPrompt: java.lang.String
-requestingUserName: java.lang.String
-reset(): void
requestPasswordAuthentication(): java.lang.Authenticator
#getRequestingSite(): java.net.InetAddress
#getRequestingPort(): int
#getRequestingProtocol(): java.lang.String
#getRequestingPrompt(): java.lang.String
#getDefaultUserName(): java.lang.String
#getPasswordAuthentication(): java.lang.Authenticator
+Authenticator()

Im zweiten Schritt kreieren wir das `javax.mail.Authenticator` Objekt oder genauer eine Instanz einer Unterklasse dieser abstrakten Klasse. Damit können wir den Benutzer auffordern ein Passwort einzugeben. Falls wir dies nicht dynamisch erledigen wollen, können wir auch einfach das Objekt auf `null` setzen. Wir werden später diesen Teil ergänzen.

## Beispiel 11 Authenticator Objekt definieren

```
Authenticator a = null;
```

# NETZWERKPROGRAMMIERUNG IN JAVA

java.lang.Object
<b>javax.mail.Session</b>
-props: java.util.Properties
-authenticator: javax.mail.Authen
-authTable: java.util.Hashtable
-debug: boolean
-providers: java.util.Vector
-providersByProtocol: java.util.H
-providersByClassName: java.util.
-addressMap: java.util.Properties
-getResources: java.lang.reflect.
-getSystemResources: java.lang.re
-defaultSession: javax.mail.Sessi
-Session(: java.util.Properties, :
+getInstance(: java.util.Properti
+getInstance(: java.util.Properti
+getDefaultInstance(: java.util.F
+getDefaultInstance(: java.util.F
+setDebug(: boolean): void
+getDebug(): boolean
+getProviders(): javax.mail.Provi
+getProvider(: java.lang.String):
+setProvider(: javax.mail.Provide
+getStore(): javax.mail.Store
+getStore(: java.lang.String): jav
+getStore(: javax.mail.URLName): j
+getStore(: javax.mail.Provider):
-getStore(: javax.mail.Provider, :
+getFolder(: javax.mail.URLName):
+getTransport(): javax.mail.Trans
+getTransport(: java.lang.String)
+getTransport(: javax.mail.URLNar
+getTransport(: javax.mail.Provié
+getTransport(: javax.mail.Addres
-getTransport(: javax.mail.Provié
-getService(: javax.mail.Provider
+setPasswordAuthentication(: java
+getPasswordAuthentication(: java
+requestPasswordAuthentication(:
+getProperties(): java.util.Prope
+getProperty(: java.lang.String):
-loadProviders(: java.lang.Class)
-loadProvidersFromStream(: java.i
-loadAddressMap(: java.lang.Classe
-pr(: java.lang.String): void

Im dritten Schritt wird eine Session Instanz gebildet. Diese verwendet die Properties und das Authenticator Objekt als Parameter.

## Beispiel 12 Session Objekt

```
Session session =  
Session.getDefaultInstance(props, a);
```

javax.mail.Service
<b>javax.mail.Store</b>
-storeListeners: java.util.Vector
-folderListeners: java.util.Vectc
#Store(: javax.mail.Session, : java
+getDefaultFolder(): javax.mail.F
+getFolder(: java.lang.String): ja
+getFolder(: javax.mail.URLName):
+getPersonalNamespaces(): javax.n
+getUserNamespaces(: java.lang.St
+getSharedNamespaces(): javax.mai
+addStoreListener(: javax.mail.ev
+removeStoreListener(: javax.mail
#notifyStoreListeners(: int, : java
+addFolderListener(: javax.mail.e
+removeFolderListener(: javax.mai
#notifyFolderListeners(: int, : jav
#notifyFolderRenamedListeners(: j

Im vierten Schritt definieren wir die Ablage, den Store, in dem die Meldungen zu finden sind. Dabei müssen wir den Provider angeben. Der Einfachheit halber beschränken wir uns im folgenden Beispiel auf POP3.

## Beispiel 13 Store Objekt

```
Store store = session.getStore("POP3");
```

*Achtung:* die `getStore()` Methode ist eine Methode von `Session`, nicht von `Store`.

Im fünften Schritt bauen wir die Verbindung zur Ablage, zum Store auf. Dies geschieht mit Hilfe der `connect()` Methode.

## Beispiel 14 Verbindungsaufbau zum Store

```
store.connect("mailhost.hta.fhz.ch",  
"<benutzername>", "<passwort>");
```

Falls Sie das `Authenticator`-Objekt definiert haben und dies ungleich `null` ist, können Sie das Passwort auf `null` setzen und damit anzeigen, dass das Passwort aus dem

`Authenticator` Objekt übernommen werden soll.

Wir sind nun mit dem Store verbunden, können also Meldungen manipulieren.

Damit sind wir bei Schritt 6 angelangt: bestimmen des Folders der INBOX. Im Fall von POP3 haben wir lediglich einen Folder, die INBOX. Im Falle von IMAP sind mehrere Folder möglich. POP3 wird also wie IMAP mit einem einzigen Folder behandelt. Die `Folder` Klasse ist zu umfangreich, um noch angezeigt zu werden. Schauen Sie bitte in der JavaDoc nach, wie umfangreich die Funktionalität dieser Klasse ist. Wir kommen später nochmals darauf zurück (beim Besprechen der einzelnen Klassen).

## Beispiel 15 INBOX Folder für POP3

```
Folder inbox = store.getFolder("INBOX");
```

Jeder Folder, den Sie so erhalten ist standardmässig geschlossen. Sie müssen ihn nun manipulieren, aber als erstes sicher öffnen! Sonst können alle Manipulationen, wie lesen, löschen ,.... nicht ausgeführt werden. Ein Folder kann als `READ_ONLY` geöffnet werden. Dann kann er nur gelesen werden, also schreiben, löschen,...ist nicht möglich.

Im siebten Schritt müssen wir also den Folder aufmachen. Dies geschieht mit Hilfe der `open()` Methode des `inbox` Objekts, oder allgemein mit der entsprechenden Methode der `Folder` Klasse.

## Beispiel 16 öffnen der INBOX

```
inbox.open(Folder.READ_ONLY);
```

Im Falle von POP3 haben wir keine weiteren inneren Folder. Daher entfällt der Schritt 8: wir brauchen keine weiteren Folder zu öffnen.

# NETZWERKPROGRAMMIERUNG IN JAVA

Im neunten Schritt können wir die Meldungen herunterladen. Als erstes betrachten wir den Fall, dass wir ohne grosse Selektion einfach alle Meldungen in ein Array laden. Dies geschieht mit der `getMessage` Methode des `Folders` (in unserem Fall dem `inbox` Objekt)

## Beispiel 17 Herunterladen der Nachrichten in ein `Message[ ]` Array

```
Message[ ] messages = inbox.getMessages();
```

Im Falle von IMAP würden die Meldungen weiterhin auf dem Server bleiben. Sie würden lediglich einen Pointer auf die Meldung erhalten.

Nun sind wir bei Schritt 10 angelangt. Dabei geht es um das Bearbeiten der Nachrichten. Die `Message` Klasse stellt Ihnen nun einige Methoden zur Verfügung, mit deren Hilfe Sie die Meldungen manipulieren können, zum Beispiel ausdrucken. Dies geschieht mit der `writeTo()` Methode der `Message` Klasse. Auch die `Message` Klasse ist sehr mächtig und das Diagramm füllt mehr als eine Seite. Sie erhalten eine brauchbare Übersicht über die Funktionalität, indem Sie JavaDoc für diese Klasse anschauen.

## Beispiel 18 Ausgabe der heruntergeladenen Meldungen

```
for (int i=0; i<messages.length; i++) {
    System.out.println("----- Nachricht " + (i+1) + "-----");
    messages[i].writeTo(System.out);
}
```

Falls wir keine andere Tätigkeiten an den Nachrichten vornehmen möchten, können wir jetzt die Folder schliessen und aufräumen. Dies ist der Inhalt des elften Schrittes.

## Beispiel 19 Schliessen aller Folder

```
inbox.close(false); //false: gelöschte Nachrichten bleiben erhalten
```

Wie oben erwähnt, könnten eventuell mehrere Folder offen sein. In diesem Fall müssen wir alle der Reihe nach schliessen, so wie wir sie aufgemacht haben.

Und nun folgt der letzte Schritt, Schritt 12, das Schliessen des Stores. Dazu stellt uns der Store die entsprechende Methode zur Verfügung.

## Beispiel 20 Schliessen des Stores

```
store.close();
```

Das Ganze schauen wir uns nun in einem einfachen Beispiel, zuerst ohne GUI an.

## Beispiel 21 Vollständiges POP3 Beispiel

```
import javax.mail.*;
import javax.mail.internet.*;
import java.util.*;
import java.io.*;

public class POP3Client {

    public static void main(String[] args) {

        Properties props = new Properties();
```

# NETZWERKPROGRAMMIERUNG IN JAVA

```
String host = "mailhost.hta.fhz.ch";
String username = "zajoller";
String password = "<mein Passwort>";
String provider = "pop3";

try {

    // POP3 Server Verbindungsaufbau
    Session session = Session.getDefaultInstance(props, null);
    Store store = session.getStore(provider);
    store.connect(host, username, password);

    // Folder oeffnen
    Folder inbox = store.getFolder("INBOX");
    if (inbox == null) {
        System.out.println("No INBOX");
        System.exit(1);
    }
    inbox.open(Folder.READ_ONLY);

    // Nachrichten lesen
    Message[] messages = inbox.getMessages();
    for (int i = 0; i < messages.length; i++) {
        System.out.println("----- Message " + (i+1)
            + " -----");
        messages[i].writeTo(System.out);
    }

    // Verbindung schliessen
    // Meldungen auf dem Server lassen
    inbox.close(false);
    store.close();

}
catch (Exception e) {
    e.printStackTrace();
}

}

public static void printMessage(Message m) throws Exception {

    // Nachrichtenheader drucken
    Address[] from = m.getFrom();
    if (from != null) {
        for (int i = 0; i < from.length; i++) {
            System.out.println("From: " + from[i]);
        }
    }

    Address[] to = m.getRecipients(Message.RecipientType.TO);
    if (to != null) {
        for (int i = 0; i < to.length; i++) {
            System.out.println("To: " + to[i]);
        }
    }

    String subject = m.getSubject();
    if (subject != null) {
        System.out.println("Subject: " + subject);
    }
}
```

# NETZWERKPROGRAMMIERUNG IN JAVA

```
Date d = m.getSentDate();
if (d != null) {
    System.out.println("Date: " + d);
}

// Leerzeile
System.out.println();

// Message Body
Object content = m.getContent();
if (content instanceof String) {
    System.out.println(content);
}
else if (content instanceof InputStream) {
    InputStream in = (InputStream) content;
    int c;
    while ((c = in.read()) != -1) System.out.write(c);
}
else {
    // MIME Nachricht?
    System.out.println("Content Type Fehler");
}
}
}
```

Die Ausgabe hängt von Ihrer Mailbox ab. Der Aufbau ist aber immer der Gleiche:

1. Informationen über den Sender (Absender und IP Adresse, falls bekannt) und Datum
2. Informationen über den Empfänger und Datum
3. Mime Version
4. Message ID
5. Datum
6. From: <Adresse>
7. To: <Adresse>
8. Subject: <Text>
9. Content-Type: ...
10. Die Meldung

## 15.7. *Passwort Authentifizierung*

Die Angabe eines Passwortes in einem Programm bietet sich höchstens für das Testen an. Besser ist sicher eine Eingabeaufforderung zur Laufzeit. Idealerweise sollte das Passwort auch nicht angezeigt werden beim Eingeben. Idealerweise sollten Benutzernamen und Passwort auch nicht als Text übermittelt werden. Das geschieht aber in der Mehrzahl aller Mail Clients. IMAP ist in der Regel etwas mehr auf Sicherheit bedacht.

Wie wir weiter vorne gesehen haben, kann man beim Öffnen der Verbindung zum Message Store ein Authentisierungsobjekt als Parameter angeben. Mit `javax.mail.Authenticator` als abstrakter Klasse, kann man einen solchen Mechanismus aufbauen:

```
public abstract class Authenticator extends Object
```

# NETZWERKPROGRAMMIERUNG IN JAVA

Wann immer der Provider einen Benutzernamen oder ein Passwort wissen muss, ruft er die Methode `getPasswordAuthentication()` Methode einer benutzerdefinierten Unterklasse der Klasse `Authenticator` auf. Diese liefert ein `PasswordAuthentication` Objekt zurück, welches die benötigte Informationen enthält:

```
protected PasswordAuthentication getPasswordAuthentication()
```

<code>java.lang.Object</code>
<b><i>javax.mail.Authenticator</i></b>
<code>-requestingSite: java.net.InetAddress</code>
<code>-requestingPort: int</code>
<code>-requestingProtocol: java.lang.String</code>
<code>-requestingPrompt: java.lang.String</code>
<code>-requestingUserName: java.lang.String</code>
<code>-reset(): void</code>
<code>requestPasswordAuthentication(): PasswordAuthentication</code>
<code>#getRequestingSite(): java.net.InetAddress</code>
<code>#getRequestingPort(): int</code>
<code>#getRequestingProtocol(): java.lang.String</code>
<code>#getRequestingPrompt(): java.lang.String</code>
<code>#getDefaultUserName(): java.lang.String</code>
<code>#getPasswordAuthentication(): PasswordAuthentication</code>
<code>+Authenticator()</code>

## Bemerkung 2 Authentication in java.net und in javax.mail

Damit die JavaMail Klassen auch in Java 1.x mit  $x < 2$  eingesetzt werden können, musste Sun die Klasse aus `java.net` mehr oder weniger duplizieren.

In `java.net` findet Sie zwei Klassen:

```
java.net.PasswordAuthentication  
java.net.Authenticator
```

die im wesentlichen die selbe Funktion wie die Authentifizierungsklasse aus `javax.mail` haben. Der Nachteil dieser Duplikation ist, dass immer wenn Sie beide Packages importieren müssen, Sie darauf achten müssen, die Klassen vollqualifiziert zu verwenden, da sonst unklar ist, aus welchem Paket die Klasse zu verwenden ist.

Nun wollen wir den Passwortschutz in unsere Programme einbauen und dazu die Authentifizierungsklassen verwenden! Swing bietet bereits eine Klasse an, mit der Passwörter sicher abgefragt werden können: mit Hilfe der Klasse `JPasswordField`.

Machen wir uns das Leben besonders einfach und verwenden wir einfach diese Klassen in einem Swing basierten GUI.

## Beispiel 22 Eine GUI Authentifizierungsklasse

```
package swinguiauthentifizierung;  
  
/**  
 * Title: SwingGUImitAuthentifizierung  
 * Description: Einfache Dialogbox  
 * Jede Menge Java Code  
 * Aber wenn es sein muss, muss es sein.  
 * Copyright: Copyright (c) 2000  
 * Company: Joller-Voss GmbH  
 * @author J.M.Joller  
 * @version 1.0  
 */  
  
import javax.mail.*;  
import javax.swing.*;  
import java.awt.*;  
import java.awt.event.*;
```

# NETZWERKPROGRAMMIERUNG IN JAVA

```
public class MailAuthenticator extends Authenticator {

    private JDialog passwordDialog = new JDialog(new JFrame(), true);
    private JLabel mainLabel = new JLabel(
        "Bitte Benutzernamen und Passwort eingeben: ");
    private JLabel userLabel = new JLabel("Benutzernamen: ");
    private JLabel passwordLabel = new JLabel("Passwort: ");
    private JTextField usernameField = new JTextField(20);
    private JPasswordField passwordField = new JPasswordField(20);
    private JButton okButton = new JButton("OK");

    public MailAuthenticator() {
        this("");
    }
    public MailAuthenticator(String username) {

        Container pane = passwordDialog.getContentPane();
        pane.setLayout(new GridLayout(4, 1));
        pane.add(mainLabel);
        JPanel p2 = new JPanel();
        p2.add(userLabel);
        p2.add(usernameField);
        usernameField.setText(username);
        pane.add(p2);
        JPanel p3 = new JPanel();
        p3.add(passwordLabel);
        p3.add(passwordField);
        pane.add(p3);
        JPanel p4 = new JPanel();
        p4.add(okButton);
        pane.add(p4);
        passwordDialog.pack();

        ActionListener al = new HideDialog();
        okButton.addActionListener(al);
        usernameField.addActionListener(al);
        passwordField.addActionListener(al);

    }
    class HideDialog implements ActionListener {

        public void actionPerformed(ActionEvent e) {
            passwordDialog.hide();
        }
    }
    public PasswordAuthentication getPasswordAuthentication() {
        passwordDialog.show();
        // getPassword() liefert ein Array, aus Sicherheitsgründen
        // Dieses muss in eine Zeichenkette umgewandelt werden.
        // PasswordAuthentication() constructor.
        String password = new String(passwordField.getPassword());
        String username = usernameField.getText();
        // Löschen des PWD
        // Der Provider kennt das PWD noch
        passwordField.setText("");
        return new PasswordAuthentication(username, password);
    }
}
```

Diese Klasse zeigt ein einfaches Dialogfenster an mit den zwei Eingabefeldern für den Benutzernamen und das Passwort. Witzigerweise speichert die Swing Klasse die Angaben in

# NETZWERKPROGRAMMIERUNG IN JAVA

einem Array ab, um die Sicherheit zu erhöhen. Damit ist es einfach, das Passwort nach Gebrauch zu überschreiben. Damit existiert das Passwort während einer kürzeren Zeit im Hauptspeicher; damit wird die Kopierbarkeit reduziert oder die Sicherheit erhöht.



Jetzt modifizieren wir unseren POP3 Client und ergänzen ihn mit der neu definierten Klasse. Damit sind wir auch in der Lage an Stelle des Nullargumentes beim Aufruf der connect() Methode ein Authentication Objekt anzugeben.

## Beispiel 23 Sicherer POP3 Client

```
package sichererpop3client;

import javax.mail.*;
import javax.mail.internet.*;
import java.util.*;
import java.io.*;

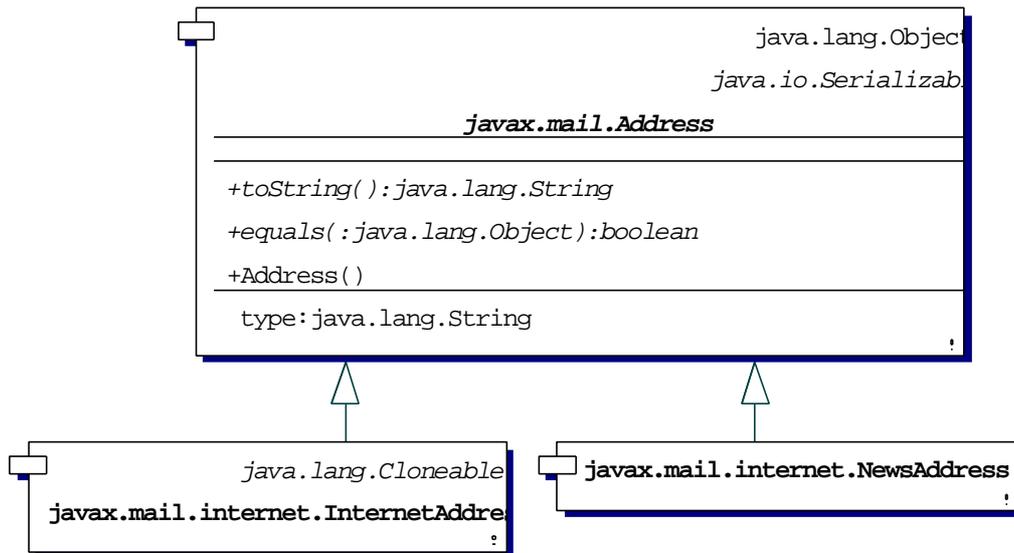
public class SecurePOP3Client {
    public static void main(String[] args) {
        Properties props = new Properties();
        String host = "mailhost.hta.fhz.ch";
        String provider = "pop3";
        try {
            //Verbindungsaufbau
            Session session = Session.getDefaultInstance(props,
                new MailAuthenticator());
            Store store = session.getStore(provider);
            store.connect(host, null, null);

            // Folder öffnen
            Folder inbox = store.getFolder("INBOX");
            if (inbox == null) {
                System.out.println("No INBOX");
                System.exit(1);
            }
            inbox.open(Folder.READ_ONLY);
            // Mail lesen
            Message[] messages = inbox.getMessages();
            for (int i = 0; i < messages.length; i++) {
                System.out.println("----- Message " + (i+1)
                    + " -----");
                messages[i].writeTo(System.out);
            }
            // Verbindungsabbau
            // Meldungen bleiben auf dem Server
            inbox.close(false);
            store.close();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        // explizites Verlassen des Programms
        System.exit(0);
    }
}
```

}

## 15.8. Adressen

Schauen wir uns als erstes eine Übersicht über die `javax.mail.Address` Klasse an, sowie deren Umfeld.



```
public abstract class Address extends Object implements Serializable {
    public abstract String getType();
    public abstract String toString();
    public abstract boolean equals(Object par1);
    public Address() {
    }
}
```

Die Adressklasse besitzt gemäss obigem Diagramm (aus TogetherJ) zwei Unterklassen, die jeweils bestimmten Diensten entsprechen, einer für SMTP und einer für NNTP (Newsgroups). Beide stellen also Erweiterungen der Oberklasse `Address` dar. Falls man weitere Dienste implementiert werden sollten, müsste man also entsprechende Unterklassen definieren.

```
public class InternetAddress extends Address
public class NewsAddress extends Address
```

Schauen wir uns die Details der einzelnen Klassen an und beginnen wir mit der Oberklasse, der Wurzel aller Adressklassen.

### 15.8.1. Die Address Klasse

Die `Address` Klasse selbst ist, wie man aus dem obigen Diagramm und der Java Beschreibung erkennen kann, sehr einfach aufgebaut. Sie verfügt lediglich über einige grundlegende Methoden, die die entsprechenden Methoden von `Object` überschreiben.

Da alle Methoden abstrakt definiert werden, müssen alle in den Unterklassen überschrieben werden. Dies hat den Nachteil, dass die Semantik der Methoden auf der Stufe dieser Klasse offen bleibt. Beispielsweise könnte die Definition "gleich" in den unterschiedlichen

# NETZWERKPROGRAMMIERUNG IN JAVA

Unterklassen auf unterschiedliche und wenig plausible Art und Weise definiert werden. Auf der andern Seite kann man durch überschreiben der `toString()` Methode eine sinnvolle Zeichenkettendarstellung definieren. Die Methode `getType()` liefert beispielsweise "rfc822".

## 15.8.2. Die `InternetAddress` Klasse

Java stellt Ihnen eine Klasse zur Verfügung, mit der Sie die Definition und Konformität einer email Adresse gemäss RFC822 kontrollieren können. Dies ist eine äusserst wertvolle Überprüfung in fast allen Mail Anwendungen.

### Beispiel 24 Gültige `InternetAddress` Objekte

1. [zajoller@hta.fhz.ch](mailto:zajoller@hta.fhz.ch)
2. [zajoller@hta.fhz.ch](mailto:zajoller@hta.fhz.ch) (Josef M. Joller)

Es ist also erlaubt, eine Textbeschreibung mitzugeben. Der Zustand eines `InternetAddress` Objekts wird durch drei Datenfelder festgehalten:

alle Datenfelder sind geschützt und Zeichenketten:

```
protected String address
personal
encodedPersonal
```

Im obigen Beispiel hätten diese drei Felder folgende Werte:

1. `address = zajoller@hta.fhz.ch`
2. `personal = Josef M. Joller`
3. `encodedPersonal =`

Im `encodedPersonal` Datenfeld steht genau dann etwas, wenn beispielsweise ein Name in einer nicht ASCII

Kapitel 15 SMTP POP3 und



# NETZWERKPROGRAMMIERUNG IN JAVA

Darstellung erfasst wurde. Dieser String entspricht einer speziellen Kodierung gemäss RFC 2047 *MIME (Multipurpose Internet Mail Extensions) Part Three : Message Header Extensions for Non ASCII Text*.

Die Klasse kennt vier Konstruktoren:

```
public class InetAddress extends Address implements Cloneable {

    protected String address;
    protected String personal;
    protected String encodedPersonal;

    private static final String rfc822phrase;
    private static final String specialsNoDotNoAt;
    private static final String specialsNoDot;

    public InetAddress() {
    }
    public InetAddress(String address) {
    }
    public InetAddress(String address, String personal) {
    }
    public InetAddress(String address, String personal, String charset)
        {
    }
}
```

Die Konstruktoren tun genau das was man von ihnen gemäss den Parametern erwarten würde.

## Beispiel 25 Konstruktion eines **Address** Objektes mit persönlichem Namen

```
Address tim = new InetAddress("zajoller@hta.fhz.ch", "Josef M. Joller");
```

Die Klasse besitzt je drei set... , get... und plus Hilfsmethoden:

```
// set...
public void setAddress(String address) {
}
public void setPersonal(String name, String charset) {
}
public void setPersonal(String name) {
}
// get...
public String getType() {
}
public String getAddress() {
}
public String getPersonal() {
}
// toString
public String toString() {
}
// equals
public boolean equals(Object par1) {
}
// hashCode
public int hashCode() {
}
// clone
```

# NETZWERKPROGRAMMIERUNG IN JAVA

```
public Object clone() {  
}
```

Die Semantik der einzelnen Methoden ist recht einfach:

1. setAddress()  
setzt das address Datenfeld auf den Parameterwert
2. setPersonal()  
setzt das personal bzw. encodedPersonal Datenfeld auf den Parameterwert
3. getAddress() liefert die Adresse;  
getPersonal() liefert das personal oder encodedPersonal Datenfeld entschlüsselt  
getType() liefert in der Regel die Zeichenkette "rfc822"
4. toString() liefert die email Adresse in einer brauchbaren Zeichenkettendarstellung,  
gemäss RFC 822, also in der Darstellung, wie sie im FROM: und im TO: auftreten sollten.
5. hashCode() und equals() funktionieren wie bei Object

Zusätzlich besitzt die Klasse fünf statische Methoden, von denen vier Adressen in und aus Zeichenketten umgewandelt werden können:

```
public static String toString(Address[] addresses) {  
public static String toString(Address[] addresses, int used) {  
public static InetAddress[] parse(String addressList) {  
public static InetAddress[] parse(String s, boolean strict) {
```

Die Semantik der einzelnen Methoden:

die toString() Methoden liefern ein Array mit Komma getrennten Adressen in reiner ASCII Darstellung. int gibt die Anzahl Zeichen an, welche der Zeichenkette vorangestellt werden, beispielsweise im To: und From: . Dadurch wird die Zeichenkette an der richtigen Stelle eingefügt.

Zwei parse() Methoden wandeln Komma getrennte Adressen in InetAddress um

```
public static InetAddress[] parse(String addresslist) {  
}  
public static InetAddress[] parse(String s, boolean strict) {  
}
```

Falls der Parameter strict false ist, werden die Adressen an Stelle von Kommas durch Leerzeichen getrennt.

Die Methode

```
static InetAddress getLocalAddress(Session session)
```

überprüft die lokalen Eigenschaften und versucht die email Adresse zu finden (lokale Eigenschaften sind: mail.user, mail.host und user.name).

## Beispiel 26 Bestimmen der email Adresse aus lokalen Eigenschaften

```
msg.setFrom(InetAddress.getLocalAddress());
```

Dabei muss man aber beachten, dass die entsprechenden Properties korrekt gesetzt wurden. Das obige Beispiel zeigt aber, wie man beispielsweise vermeiden könnte ein Passwort fix einzugeben. Bekanntlich kann man Properties auch auf der Kommandozeile spezifizieren. Damit wird das Passwort erst zur Laufzeit spezifiziert.

## 15.8.3. Die `NewsAddress` Klasse

Mit einem entsprechenden Provider kann man aus `JavaMail` auch auf Newsgroups zugreifen. Das API ist im wesentlichen gleich aufgebaut wie jenes von POP oder IMAP Mailboxen. Anstelle von `InternetAddress` verwendet man einfach `NewsAddress`:

```
public class NewsAddress extends Address
```

Ein `NewsAddress` Objekt repräsentiert eine Usenet Newsgruppe, wie beispielsweise `comp.lang.java.machine`. Die Adresse kann auch noch den Hostnamen umfassen, der Name des News-Servers. Der Zustand des `NewsAddress` Objekts wird durch die zwei Datenfelder

```
protected String newsgroup  
protected String host
```

festgelegt. Das Datenfeld `newsgroup` enthält den Namen der Newsgroup - beispielsweise, `comp.lang.java.machine`. Das `host` Datenfeld ist entweder null oder enthält den Hostnamen des News Server - beispielsweise `news.bluewin.ch`. Beide Datenfelder werden im Konstruktor gesetzt. Die Klasse kennt drei Konstruktoren:

```
public NewsAddress()  
public NewsAddress(String newsgroup)  
public NewsAddress(String newsgroup, String host)
```

Die Funktionsweise ist genau die, die man erwartet:

```
Address java_machine = new  
    NewsAddress("comp.lang.java.machine", "news.hta.fhz.ch");
```

Die Klasse besitzt acht Objektmethoden (drei zum bestimmen von Attributen [`get...`], zwei zum Setzen von Attributen [`set...`] und drei Hilfsmethoden):

```
public String    getType()  
public String    getHost()  
public String    getNewsgroup()  
public void      setNewsgroup(String newsgroup)  
public void      setHost(String host)  
public String    toString()  
public boolean   equals(Object o)  
public int       hashCode()
```

Die `setNewsgroup()` und `setHost()` Methoden setzen die Newsguppe und den Host des Objekts. Die `getNewsgroup()` und `getHost()` Methoden liefern die Werte der Newsguppe und des Host Datenfeldes. Die `getType()` Methode liefert die Zeichenkette "news".

Die `toString()` Methode liefert den Namen der Newsguppe in Form einer Newsguppe: die Kopfzeile einer Usenet Nachricht. Die `equals()` und `hashCode()` Methoden haben die übliche Bedeutung.

# NETZWERKPROGRAMMIERUNG IN JAVA

Neben den Objekt basierten Methoden verfügt die Klasse auch noch über Klassenmethoden:

```
public static String toString(Address[] addresses)
public static NewsAddress[] parse(String newsgroups)
```

Diese `toString` Methode konvertiert ein Datenfeld, ein Array, von `Address` Objekten in eine durch Kommas getrennte Liste mit Newsgruppen Namen.

Die `parse()` Methode macht genau das Umgekehrte: sie wandelt eine durch Komma getrennte Liste von Zeichenketten mit Newsgruppennamen wie "comp.lang.java.programmer, comp.lang.java.gui.comp.lang.java.help" in ein Array von `NewsAddress` Objekten.

Sun's Referenzimplementation von `JavaMail` enthält keinen Serviceprovider für News. Daher müssten Sie einen News-Provider installieren, beispielsweise einen aus der Liste weiter vorne. Sie finden auch Hinweise auf Lösungen in den FAQs bei `jGuru`: <http://www.jguru.com/jguru/faq/faqpage.jsp?name=JavaMail> bei dem Sie auch Hinweise und die Beantwortung vieler Standardfragen zu anderen Java Themen finden.

## 15.9. Die URLName Klasse

Das Paket `javax.mail.URLName` stellt den Namen einer URL dar. Die URL wird als Zeichenkette dargestellt, es wird also keine Verbindung zu einem Rechner aufgebaut. URL Namen dienen einfach der Beschreibung der im System vorhandenen Objekte und Klassen.

```
public class URLName extends Object
```

Die Methoden von `URLName` sind analog zu jenen der Klasse `java.net.URL`. Die Ausnahme ist, wie bereits oben erwähnt, dass hier keine Verbindung aufgebaut wird. Aber auch hier haben wir diverse Methoden zur Verfügung, mit deren Hilfe der URL String zerlegt und analysiert werden kann.

### 15.9.1. Die Konstruktoren

Die Klasse kennt drei Konstruktoren für `URLName`. Die Signatur der Konstruktoren zeigt deren unterschiedliche Einsatzmöglichkeiten.

```
public URLName(String protocol, String host, int port, String file,
                String userName, String password)
public URLName(URL url)
public URLName(String url)
```

Der Konstruktor verlangt nicht, dass auch ein Protokollhandler zur Verfügung steht, da ja lediglich eine Zeichenkette manipuliert wird, keine Verbindung aufgebaut wird. Wir könnten also problemlos folgende Adressen eingeben:

```
pop3://<benutzername>:<passwort>@mailhost.hta.fhz.ch/INBOX
```

Damit können wir auf eine bestimmte INBOX verweisen.

### 15.9.2. Methoden zum Parsen eines URLName

Diese Methoden sind eigentlich die wichtigsten für diese Klasse:

```
public int      getPort()
public String   getProtocol()
```

# NETZWERKPROGRAMMIERUNG IN JAVA

```
public String    getFile()
public String    getRef()
public String    getHost()
public String    getUsername()
public String    getPassword()
```

Die Bedeutung dieser Methoden ergibt sich aus dem Namen und aus Analogie zur `java.net.URL` Klasse. Alle Klassen liefern ein null Objekt, falls die entsprechende Angabe in der Zeichenkette fehlt, ausser beim Port: diese Methode liefert -1 in diesem Fall.

Damit man die Beziehung zur Klasse `java.net.URL` herstellen kann, wird auch eine entsprechende Methode zur Verfügung gestellt. Falls der entsprechende Protokollhandler nicht existiert, wird eine `Exception`, `MalformedURLException` geworfen.

```
public URL getURL() throws MalformedURLException
```

Und schliesslich hier noch die drei Hilfsmethoden:

```
public boolean   equals(Object o)
public int       hashCode()
public String    toString()
```

beispielsweise zum Umwandeln einer URL in die entsprechende Zeichenketten-Darstellung.

Als Anwendung dieser Klasse betrachten wir einen universellen Mail Client. Alle Informationen über Protokoll, Host, ... wird als URL übergeben.

## Beispiel 27 Protokoll unabhängiger Mail Client

```
package protokollunabhaengigermailclient;

/**
 * Title:      Einfacher MailClient mit URLName als Parameter
 * Description: Universeller Client, der einen URLName String als
 *              Parameter übernimmt
 * Copyright:   Copyright (c) 2000
 * Company:    Joller-Voss GmbH
 * @author J.M.Joller
 * @version 1.0
 */
import javax.mail.*;
import javax.mail.internet.*;
import java.util.*;
import java.io.*;

public class MailClient {

    public static void main(String[] args) {
        if (args.length == 0) {
            System.err.println(
                "Usage: java MailClient
protocol://username:password@host:port/foldername");
            //pop3://<<username>><<password>>@pop.bluewin.ch/inbox
            return;
        }

        URLName server = new URLName(args[0]);
        //URLName server = new URLName(temp);
```

# NETZWERKPROGRAMMIERUNG IN JAVA

```
System.out.println("URLName festgelegt");

try {
    System.out.println("Verbindungsaufbau");
    Session session = Session.getDefaultInstance(new Properties(), null);

    // Verbindung zum Server aufbauen und Folder öffnen
    Folder folder = session.getFolder(server);
    System.out.println("Verbindungsaufbau abgeschlossen");
    System.out.println("Folder oeffnen");
    if (folder == null) {
        System.out.println("Folder " + server.getFile() + " nicht
gefunden.");
        System.exit(1);
    }
    folder.open(Folder.READ_ONLY);
    System.out.println("Folder geoeffnet");

    // Lesen der Messages
    System.out.println("Lesen der Messages");
    Message[] messages = folder.getMessages();
    System.out.println("Messages gelesen");
    for (int i = 0; i < messages.length; i++) {
        System.out.println("----- Message " + (i+1)
+ " -----");
        messages[i].writeTo(System.out);
    }

    // Abbau der Verbindung
    // Meldungen bleiben auf dem Server
    folder.close(false);
}
catch (Exception e) {
    e.printStackTrace();
}
}
```

URLName festgelegt  
Verbindungsaufbau  
Verbindungsaufbau abgeschlossen  
Folder oeffnen  
Folder geoeffnet  
Lesen der Messages  
Messages gelesen

```
----- Message 1 -----
Return-Path: <JJoller@hta.fhz.ch>
Received: from mta1n.bluewin.ch (172.21.1.230) by mss1n.bluewin.ch (Bluewin AG MSS engine 5.1.053)
id 3A1BAB8700ABFF29 for fam_joller@bluewin.ch; Mon, 11 Dec 2000 11:48:35 +0100
Received: from ntindex.ztl.ch (147.88.1.197) by mta1n.bluewin.ch (Bluewin AG MX engine 5.1.053)
id 3A2F9F9D001AB178 for fam_joller@bluewin.ch; Mon, 11 Dec 2000 11:49:35 +0100
Received: by ztnt05.ztl.ch with Internet Mail Service (5.5.2650.21)
id <XQPC4BDC>; Mon, 11 Dec 2000 11:49:37 +0100
Received: from mercury.Sun.COM ([192.9.25.1]) by ntindex.ztl.ch with SMTP (Microsoft Exchange Internet
Mail Service Version 5.5.2650.21)
id XQPC4BDB; Mon, 11 Dec 2000 11:49:33 +0100
Received: from ezrh04.Swiss.Sun.COM ([129.156.51.2])
```

# NETZWERKPROGRAMMIERUNG IN JAVA

by mercury.Sun.COM (8.9.3+Sun/8.9.3) with ESMTP id CAA01389  
for <jjoller@hta.fhz.ch>; Mon, 11 Dec 2000 02:49:29 -0800 (PST)  
Received: from giornico (giornico [129.156.52.94])  
by ezrh04.Swiss.Sun.COM (8.8.8+Sun/8.8.8/ENSMail,v2.0) with SMTP id LAA12887;  
Mon, 11 Dec 2000 11:49:27 +0100 (MET)  
Message-Id: <200012111049.LAA12887@ezrh04.Swiss.Sun.COM>  
Date: Mon, 11 Dec 2000 11:49:27 +0100 (MET)  
From: Sandra Berger - Sun Switzerland Zurich - Educational Telesales <Sandra.Berger@swiss.sun.com>  
Reply-To: Sandra Berger - Sun Switzerland Zurich - Educational Telesales <Sandra.Berger@swiss.sun.com>  
Subject: Zertifizierungen: Schulungen von Sun Educational Services  
To: jjoller@hta.fhz.ch  
Cc: Sandra.Berger@swiss.sun.com  
MIME-Version: 1.0  
Content-Type: TEXT/plain; charset=ISO-8859-1  
Content-Transfer-Encoding: QUOTED-PRINTABLE  
Content-MD5: uzzRr8jNNMhzhg19w0yW/g==  
X-Mailer: dtmail 1.3.0 @(#)CDE Version 1.3.2 SunOS 5.7 sun4u sparc  
> -----  
> Von: =09Sandra Berger - Sun Switzerland Zurich - Educational Telesales[SM=  
TP:SANDRA.BERGER@SWISS.SUN.COM]  
> Gesendet: =09Montag, 11. Dezember 2000 11:49:27  
> An: =09jjoller@hta.fhz.ch  
> Cc: =09Sandra.Berger@swiss.sun.com  
> Betreff: =09Zertifizierungen: Schulungen von Sun Educational Services  
> Diese Nachricht wurde automatisch von einer Regel weitergeleitet.  
>=20  
Sehr geehrter Herr Joller=20  
...

Mit Hilfe des URL Namens werden die Angaben aus dem Programm auf die Kommandozeile verschoben. Ganz nettes Beispiel für den praktischen Einsatz von `URLName`. Beachten Sie folgende Punkte:

1. die Angabe des Passwortes erfolgt im URL
2. der Mail Host, nicht der Domain Name muss angegeben werden (POP3 Host)
3. als Folder können Sie INBOX angeben

## 15.10. Die Message Klasse

Die `javax.mail.Message` Klasse ist eine Oberklasse für alle individuellen emails, news, und ähnliche Meldungen, wie man auch aus JavaDoc erkennen kann:

```
public abstract class Message extends Object implements Part
```

Als einzige konkrete Unterklasse von `Message` wurde `javax.mail.internet.MimeMessage` definiert. Mit diesem `Message` Typ kann man sowohl Mail als auch News Meldungen behandeln. Zusätzliche Formate müssen durch die Provider implementiert werden.

Die `Message` Klasse definiert im wesentlichen `get...` und `set...` Methoden, mit deren Hilfe allgemeine Eigenschaften einer Mail Message definiert werden, beispielsweise Adressen, Empfänger, Betrifft, Inhalt der Meldung usw. also sozusagen ein Umschlag für alle möglichen Nachrichten.

Zusätzlich implementiert `Message` auch noch das `Part` Interface. Das `Part` Interface befasst sich vor allem mit dem Rumpf, dem Body der Nachricht. Mit den Methoden der das Interface `Part` implementierenden Klassen kann man den Inhaltstyp setzen oder den Inhalt selbst,

# NETZWERKPROGRAMMIERUNG IN JAVA

Header Informationen ergänzen und vieles mehr. Speziell bei emails, die aus mehreren Anteilen bestehen, sind diese Methoden sehr hilfreich. Auch Anhänge, Attachments, werden mit Hilfe von Methoden dieser Klassen behandelt.

Als erstes diskutieren wir `Message`, dann `Part`.

## 15.10.1. Kreieren von Nachrichten / Meldungen / Messages

Die Klasse `Message` besitzt drei Konstruktoren:

```
protected Message()  
protected Message(Folder folder, int messageNumber)  
protected Message(Session session)
```

Alle Konstruktoren sind `protected`, also nur in Unterklassen sinnvoll einsetzbar. Falls man eine generelle Meldung versendet, wird man eine der konkreten Unterklassen, beispielsweise `MimeMessage` einsetzen. Falls man Nachrichten liest, dann stehen vermutlich `Session` oder `Folder` Objekte zur Verfügung und man wird den entsprechenden Konstruktor einsetzen.

### 15.10.1.1. Nachrichten beantworten

Sofern Sie bereits eine Meldung haben, können Sie mit Hilfe der `reply()` Methode auf die bestehende Nachricht antworten:

```
public abstract Message reply(boolean replyToAll) throws MessagingException
```

Diese Methode kreiert eine neue Meldung, ein neues Objekt, wobei der "Betreff" Eintrag übernommen wird. Falls die Boole'sche Variable `replyToAll` wahr ist, wird die Antwort an alle Empfänger der Originalnachricht gesandt. Die Originalmeldung erscheint in der Antwort nicht mehr. Falls dies gewünscht wird, muss dies von Hand implementiert werden.

### 15.10.1.2. Nachrichten aus Foldern lesen

Beim Lesen von Nachrichten werden `Message` Objekte kreiert. Dies geschieht mit Hilfe der `getMessage()` und `getMessages()` Methoden:

```
public abstract Message getMessage(int messageNumber) throws...  
public Message[ ] getMessages(int start, int end) throws ...  
public Message[ ] getMessages(int[ ] messageNumbers) throws ...  
public Message[ ] getMessages() throws MessagingException
```

Die ersten drei Methoden erlauben es dem Aufrufenden die Meldungen zu spezifizieren, die gelesen werden sollen. Die letzte Methode liest einfach alle Meldungen.

## 15.10.2. Header Info

Eine typische RFC 822 Nachricht enthält einen Header, der etwa folgendermassen aussieht:

```
....  
Message-Id: <200012111049.LAA12887@ezrh04.Swiss.Sun.COM>  
Date: Mon, 11 Dec 2000 11:49:27 +0100 (MET)  
From: Sandra Berger - Sun Switzerland Zurich - Educational Telesales <Sandra.Berger@swiss.sun.com>  
Reply-To: Sandra Berger - Sun Switzerland Zurich - Educational Telesales <Sandra.Berger@swiss.sun.com>  
Subject: Zertifizierungen: Schulungen von Sun Educational Services  
To: jjoller@hta.fhz.ch  
Cc: Sandra.Berger@swiss.sun.com  
MIME-Version: 1.0
```

# NETZWERKPROGRAMMIERUNG IN JAVA

Content-Type: TEXT/plain; charset=ISO-8859-1  
Content-Transfer-Encoding: QUOTED-PRINTABLE  
Content-MD5: uzzRr8jNNMhzhg19w0yW/g==  
X-Mailer: dtmail 1.3.0 @(#)CDE Version 1.3.2 SunOS 5.7 sun4u sparc  
...

Das Beispiel ist eine Kopie der Meldung weiter vorne.

Die Felder können sich je nach Mailserver unterscheiden. Aber zumindest werden folgende Felder vorhanden sein:

- From:
- To:
- Date:
- Subject:

Oft werden folgende Felder vorhanden sein:

- cc: (carbon copy)
- bcc: (blind carbon copy: Kopie geht an diese Adresse, ohne dass dies angezeigt wird)

Diese Felder sind in der Regel über get... und set... Methoden erreichbar oder manipulierbar.

## 15.10.2.1. Die From Adresse

Die folgende vier Methoden gestatten den Zugriff (get... und set...) auf das From: Feld einer Meldung:

```
public abstract Address[] getFrom() throws MessagingException
public abstract void setFrom() throws ...,IllegalWriteException, ...
public abstract void setFrom(Address address) throws...,...,IllegalStateExc
public abstract void addFrom(Address[ ] addresses) throws .....,.....
```

Die `getFrom()` Methode liefert ein Datenfeld (Array) von `Address` Objekten, je ein Objekt für jede vorkommende Adresse im From: Header. In der Regel wird die Meldung kaum von mehr als einem Sender stammen! Falls der From Teil in der Nachricht fehlt, wird ein `null` Objekt zurück geliefert.

Die `setFrom()` und `addFrom()` Methoden ohne Argumente setzen die entsprechenden Felder einfach auf die in `mail.user` Property gesetzten Werte. Falls diese nicht vorliegt, wird versucht an deren Stelle `mail.name` zu verwenden. Falls eine `setFrom(...)` Methode mit Argument aufgerufen wird, wird die entsprechende Adresse gesetzt.

## 15.10.2.2. Die ReplyTo Adresse

Einige Meldungen enthalten ein ReplyTo Feld. Dies gibt an, an welche Adresse allfällige Nachrichten zurück gesandt werden sollten. Die Klasse dafür stellt zwei Methoden zur Verfügung:

```
public Address[] getReplyTo() throws MessagingException
public void setReplayTo(Address[ ] addresses) throws .....,.....
```

Die Bedeutung der Methoden ergibt sich aus den obigen.

# NETZWERKPROGRAMMIERUNG IN JAVA

## 15.10.2.3. Die Recipient Adresse

Da der Empfänger einer Nachricht sich in einem der Felder To:, Cc: Bcc: befinden kann, ist es etwas umständlich herauszufinden, wo jetzt der Empfänger steht. Die Klasse bietet eine "klassifizierte" Methode, mit deren Hilfe dieses Problem gelöst werden kann:

```
Message.RecipientType.TO  
Message.RecipientType.CC  
Message.RecipientType.BCC
```

Diese können mit Hilfe zweier Methoden gefunden werden:

```
public abstract Address[ ] getRecipients(Message.RecipientType type) ...  
public Address[ ] getAllRecipients() throws MessagingException
```

Die erste Methode liefert ein Array von Adressen, eines pro Adresse im angegebenen Header. Falls keine gefunden wird, wird das null Objekt zurück geliefert. Falls der Header falsch formatiert ist, wird eine Ausnahme geworfen. Die `getAllRecipients()` Methode liefert alle in den To:, Cc: und Bcc: vorhandenen Adressen.

Die folgenden Methoden setzen den Empfänger:

```
public abstract void setRecipients(Message, RecipientType type,  
    Address[ ] addresses) throws .....  
public void setRecipient(Message.RecipientType type, Address address) ...  
public abstract void addRecipients(Message.RecipientType type,  
    Address[ ] addresses) .....  
public void addRecipient(Message.RecipientType type, Address address) ...
```

Alle vier Methoden können eine `MessagingException` werfen, typischerweise weil das Format einer der Adressen nicht korrekt ist.

## 15.10.2.4. Das Subject einer Nachricht

Dieses Feld ist eine einfache Zeichenkette. Die Klasse stellt zwei Methoden zur Verfügung, eine zum Setzen und eine zum Lesen:

```
public abstract String getSubject() throws MessagingException  
public abstract void setSubject(String subject) throws .....
```

Wie zuvor wird das null Objekt zurück geliefert, falls das Subject nicht definiert ist.

## 15.10.2.5. Datum der Nachricht

Jede Meldung enthält auch ein Datum, an dem die Meldung gesendet und empfangen wurde. Die folgenden Methoden liefern oder setzen diese Informationen:

```
public abstract Date getSentDate() throws MessagingException  
public abstract void setSentDate(Date date) ..., ..., ...  
public abstract Date getReceivedDate() ...
```

Die Implementation dieser Methoden muss die Datumsinformation in der korrekten Java Formattierung liefern bzw. umwandeln.

Das folgende Beispiel illustriert einige dieser Methoden an Hand eines einfachen Beispiels, wobei wir lediglich die Headerinformation ausdrucken werden.

# NETZWERKPROGRAMMIERUNG IN JAVA

Zur Authentifizierung verwenden wir die Dialog Box aus Swing. Die Ausgabe erfolgt in die Standardausgabe:

## Beispiel 28 Lesen der Header Informationen

```
package readmailheader;

/**
 * Title:      Lesen von Mail Header Informationen
 * Description: Einfacher Mail Client, der die Header Informatione liest.
Diese werden angezeigt. Das Programm unterscheidet sich kaum von den
bisherigen, bis auf die Einschränkung auf die Headerinfo.
 * Copyright:  Copyright (c) 2000
 * Company:    Joller-Voss GmbH
 * @author J.M.Joller
 * @version 1.0
 */
import javax.mail.*;
import javax.mail.internet.*;
import java.util.*;

public class HeaderComponent {

    public static void main(String[] args) {

        if (args.length == 0) {
            System.err.println(
                "Usage: java HeaderComponent
protocol://username@host:port/foldername");
            return;
        }

        URLName server = new URLName(args[0]);

        try {

            System.out.println("Session wird aufgebaut");
            Session session = Session.getDefaultInstance(new Properties(),
                new MailAuthenticator(server.getUsername()));
            System.out.println("Session aufgebaut");

            // Verbindungsaufbau
            // Öffnen des Folders
            System.out.println("Folder wird bestimmt");
            Folder folder = session.getFolder(server);
            if (folder == null) {
                System.out.println("Folder " + server.getFile() + " not found.");
                System.exit(1);
            }
            folder.open(Folder.READ_ONLY);
            System.out.println("Folder zum Lesen geöffnet");

            // Lesen der Meldungen
            System.out.println("Lesen der Meldungen");
            Message[] messages = folder.getMessages();
            System.out.println("Ausgabe der Header Informationen");
            for (int i = 0; i < messages.length; i++) {
                System.out.println("----- Message " + (i+1)
                    + " -----");
            }
        }
    }
}
```

# NETZWERKPROGRAMMIERUNG IN JAVA

```
// Here's the big change...
String from = InetAddress.toString(messages[i].getFrom());
if (from != null) System.out.println("From: " + from);
String replyTo = InetAddress.toString(
    messages[i].getReplyTo());
if (replyTo != null) System.out.println("Reply-to: "
    + replyTo);
String to = InetAddress.toString(
    messages[i].getRecipients(Message.RecipientType.TO));
if (to != null) System.out.println("To: " + to);
String cc = InetAddress.toString(
    messages[i].getRecipients(Message.RecipientType.CC));
if (cc != null) System.out.println("Cc: " + cc);
String bcc = InetAddress.toString(
    messages[i].getRecipients(Message.RecipientType.BCC));
if (bcc != null) System.out.println("Bcc: " + to);
String subject = messages[i].getSubject();
if (subject != null) System.out.println("Subject: " + subject);
Date sent = messages[i].getSentDate();
if (sent != null) System.out.println("Sent: " + sent);
Date received = messages[i].getReceivedDate();
if (received != null) System.out.println("Received: " + received);

System.out.println();
}

// Schliessen der Verbindung
// Meldungen bleiben auf dem Server
folder.close(false);

}
catch (Exception e) {
    e.printStackTrace();
}

// Abbruch
// mit exit()
System.exit(0);

}

}
```

Session wird aufgebaut

Session aufgebaut

Folder wird bestimmt

Folder zum Lesen geöffnet

Lesen der Meldungen

Ausgabe der Header Informationen

----- Message 1 -----

From:

Sandra Berger - Sun Switzerland Zurich - Educational Telesales <Sandra.Berger@swiss.sun.com>

Reply-to:

Sandra Berger - Sun Switzerland Zurich - Educational Telesales <Sandra.Berger@swiss.sun.com>

To: jjoller@hta.fhz.ch

Cc: Sandra.Berger@swiss.sun.com

Subject: Zertifizierungen: Schulungen von Sun Educational Services

Sent: Mon Dec 11 11:49:27 GMT+01:00 2000

----- Message 2 -----

From: Adrian Jetzer <wwwaci@tic.ch>

# NETZWERKPROGRAMMIERUNG IN JAVA

Reply-to: Adrian Jetzer <wwwaci@tic.ch>  
To: Ralph Rickenbach <ralph.rickenbach@artificial-life.com>  
Cc: "Josef M. Joller" <fam\_joller@bluewin.ch>  
Subject: Sitzung 21.12.00  
Sent: Tue Dec 12 11:34:33 GMT+01:00 2000

----- Message 3 -----

From: Joller\_Voss Uster <efx3874@bluewin.ch>  
Reply-to: Joller\_Voss Uster <efx3874@bluewin.ch>  
To: fam\_joller@bluewin.ch  
Subject: Faxabsender: Joller\_Voss Uster<efx3874@bluewin.ch>  
Sent: Tue Dec 12 12:24:26 GMT+01:00 2000

----- Message 4 -----

From: Abplanalp Peter <Peter.Abplanalp@FHSO.CH>  
Reply-to: Abplanalp Peter <Peter.Abplanalp@FHSO.CH>  
To: \_FHSO\_Alle\_Angestellten <FAA@FHSO.CH>  
Subject: Kantonsrat beschliesst 2% Lohnerhöhung  
Sent: Tue Dec 12 16:47:12 GMT+01:00 2000

## Bemerkung 3 Received Datum

Wie Sie im obigen Beispiel erkennen können fehlt das Receive Datum. Dies liegt daran, dass dieses Ddatum vom empfangen Server hinzugefügt werden müsste; und POP3 Server tun dies nicht!

### 15.10.2.6. Speichern der Änderungen

Einige Systeme speichern Änderungen, die mit set... gemacht wurden sofort ab. Einige lassen dies aber sein. In diesem Fall muss von Hand die Änderung gespeichert werden. Dies geschieht mit Hilfe der Methode;

```
public abstract void saveChanges() throws MessagingException, Illegal.....
```

Die Änderungen werden aber in der Regel erst geschrieben wenn der Folder geschlossen wird.

### 15.10.3. Flags

Viele Mailprogramme erlauben es dem Benutzer Zusatzinformationen abzuspeichern. Beispielsweise ob eine Nachricht gelesen wurde. Dies geschieht in JavaMail mit Hilfe von *Flags*, welche Instanzen der Klasse `javax.mail.Flags` sind:

```
public class Flags extends Object implements Cloneable
```

Sieben Flags sind vordefiniert. Die Flags gehören zur inneren Klasse `Flags.Flag`:

```
Flags.Flag.ANSWERED  
Flags.Flag.DELETED  
Flags.Flag.DRAFT  
Flags.Flag.FLAGGED  
Flags.Flag.RECENT  
Flags.Flag.SEEN  
Flags.Flag.USER
```

# NETZWERKPROGRAMMIERUNG IN JAVA

Zudem können Implementationen weitere, eigenen Flags definieren. Falls dies der Fall ist, wird das USER Flag gesetzt. Die `getFlags()` Methode liefert die Flags einer bestimmten Nachricht:

```
public abstract Flags getFlags() throws MessagingException
```

Die `isSet()` Methode testet, ob ein Flag gesetzt ist.

```
public boolean isSet(Flags.Flag flag) throws MessagingException
```

Die Flags werden analog zu oben mit `setFlag()` bzw. `setFlags()` gesetzt.

## Beispiel 29 Setzen des DELETED Flags

```
message.setFlag(Flags.Flag.DELETED true);
```

Die Meldung wird nur markiert, nicht gelöscht!

## Beispiel 30 Flags setzen und lesen

```
package flagssetzenundlesen;

/**
 * Title:          Setzen und Lesen von Message Flags
 * Description:    einfache Demo des Einsatzes von Flags
 * Copyright:      Copyright (c) 2000
 * Company:       Joller-Voss GmbH
 * @author J.M.Joller
 * @version 1.0
 */

import javax.mail.*;
import javax.mail.internet.*;
import java.util.*;

public class FlagsClient {

    public static void main(String[] args) {

        if (args.length == 0) {
            System.err.println(
                "Usage: java FlagsClient protocol://username@host:port/foldername");
            return;
        }

        URLName server = new URLName(args[0]);

        try {

            Session session = Session.getDefaultInstance(new Properties(),
                new MailAuthenticator(server.getUsername()));

            // Verbindungsaufbau zum Server und öffnen der Folder
            Folder folder = session.getFolder(server);
            if (folder == null) {
                System.out.println("Folder " + server.getFile() + " nicht
gefunden.");
            }
        }
    }
}
```

# NETZWERKPROGRAMMIERUNG IN JAVA

```
    System.exit(1);
}
folder.open(Folder.READ_ONLY);

// Nachrichten vom Server lesen
Message[] messages = folder.getMessages();
for (int i = 0; i < messages.length; i++) {
    System.out.println("----- Message " + (i+1)
        + " -----");
    // Headers bestimmen
    String from = InternetAddress.toString(messages[i].getFrom());
    if (from != null) System.out.println("From: " + from);
    String replyTo = InternetAddress.toString(
        messages[i].getReplyTo());
    if (replyTo != null) System.out.println("Reply-to: "
        + replyTo);
    String to = InternetAddress.toString(
        messages[i].getRecipients(Message.RecipientType.TO));
    if (to != null) System.out.println("To: " + to);
    String cc = InternetAddress.toString(
        messages[i].getRecipients(Message.RecipientType.CC));
    if (cc != null) System.out.println("Cc: " + cc);
    String bcc = InternetAddress.toString(
        messages[i].getRecipients(Message.RecipientType.BCC));
    if (bcc != null) System.out.println("Bcc: " + to);
    String subject = messages[i].getSubject();
    if (subject != null) System.out.println("Subject: " + subject);
    Date sent = messages[i].getSentDate();
    if (sent != null) System.out.println("Sent: " + sent);
    Date received = messages[i].getReceivedDate();
    if (received != null) System.out.println("Received: " + received);

    // Flags testen:
    if (messages[i].isSet(Flags.Flag.DELETED)) {
        System.out.println("Deleted");
    }
    if (messages[i].isSet(Flags.Flag.ANSWERED)) {
        System.out.println("Answered");
    }
    if (messages[i].isSet(Flags.Flag.DRAFT)) {
        System.out.println("Draft");
    }
    if (messages[i].isSet(Flags.Flag.FLAGGED)) {
        System.out.println("Marked");
    }
    if (messages[i].isSet(Flags.Flag.RECENT)) {
        System.out.println("Recent");
    }
    if (messages[i].isSet(Flags.Flag.SEEN)) {
        System.out.println("Read");
    }
    if (messages[i].isSet(Flags.Flag.USER)) {
        // User Flags
        // Array
        String[] userFlags = messages[i].getFlags().getUserFlags();
        for (int j = 0; j < userFlags.length; j++) {
            System.out.println("User flag: " + userFlags[j]);
        }
    }
}

System.out.println();
}
```

# NETZWERKPROGRAMMIERUNG IN JAVA

```
// Verbindungsabbau
// Nachrichten bleiben auf dem Server
folder.close(false);

}
catch (Exception e) {
    e.printStackTrace();
}

// Abschluss der Applikation
// exit()
System.exit(0);
}
}
```

Da Bluewin die Flags nicht setzt, sind bei meiner Ausgabe keine Flags angezeigt worden.

## 15.10.4. Folders

In der Regel werden Meldungen, die aus dem Internet empfangen werden, in einen Folder abgespeichert. Falls der Folder nicht bekannt ist, kann er mit der Methode `getFolder()` bestimmt werden (die Methode gehört zur `Message` Klasse, bezieht sich also auf eine Meldung):

```
public Folder getFolder()
```

Diese Methode liefert `null`, falls die Meldung in keinem Folder enthalten ist. Die Nachrichten sind einfach durchnummeriert, wie Sie sicher in den Beispielen bereits gemerkt haben. Die erste Meldung ist auch Nummer 1, nicht 0! Falls Sie die relative Position einer Nachricht bestimmen wollen, können Sie dies mit Hilfe der Methode:

```
public int getMessageNumber()
```

Alle Nachrichten, die keinem Folder angehören, haben die Nummer 0.

Service Providern steht auch noch die entsprechende `set...` Method ezur Verfügung:

```
protected void setMessageNumber(int number)
```

## 15.10.5. Suchen

Schliesslich verfügt die `Message` Klasse auch noch über eine Methode `match()` mit der Nachrichten gesucht werden können, die bestimmte Suchkriterien erfüllen:

```
public boolean match(SearchTerm term) throws MessagingException
```

## 15.11. *Das Part Interface*

Das `Part` Interface wird sowohl von `Message` als auch `BodyPart` implementiert. Jede `Message` ist also ein `Part`. Aber es gibt `Parts`, welche andere `Parts` enthalten können. Das `Part` Interface definiert drei Methodentypen:

# NETZWERKPROGRAMMIERUNG IN JAVA

- Methoden, mit denen Attribute gesetzt und gelesen werden können
- Methoden mit denen Header Informationen gesetzt und gelesen werden können
- Methoden, mit denen Inhalte gesetzt oder gelesen werden können

## 15.11.1. Attribute

JavaMail definiert fünf Attribute für Parts:

- *Size*  
die ungefähre Anzahl Bytes imPart
- *LineCount*  
die Anzahl Zeilen im Part
- *Disposition*  
eine kurze Textzusammenfassung (attachment, inline)
- *Description*  
eine kurze Textzusammenfassung des Parts
- *Filename*  
Dateinamen der Datei, von der das Anhängsel stammt (ohne Laufwerk, Server)

Einige der Attribute sind nur bedingt vorhanden, beispielsweise nur falls auch ein Attachment vorhanden ist. Alle Attribute kann man mit Hilfe einer get... Methode bestimmen:

```
public int      getSize() throws MessagingException
public int      getLineCount() throws ...
public String   getDisposition() throws ...
public String   getDescription() throws ...
public String   getFileName() throws ...
```

Die get... Methoden liefern null oder -1 falls die Information fehlt. Dazu gibt es entsprechende set... Methoden:

```
public void setDisposition(String disposition) throws ...
public void setFileName(String filename) ...
public void setDescription(String description) ...
```

Die Methode `setDisposition()` bestimmt zuerst, ob der Part als Attachment oder inline geliefert werden soll.

### Beispiel 31 Lesen von Mail Attributen

```
package lesenvonmailattributen;

/**
 * Title:      Lesen der Mail Attribute
 * Description:
 * Copyright:   Copyright (c) 2000
 * Company:    Joller-Voss GmbH
 * @author J.M.Joller
 * @version 1.0
 */

import javax.mail.*;
import javax.mail.internet.*;
import java.util.*;
```

```
public class AttributeClient {

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

# NETZWERKPROGRAMMIERUNG IN JAVA

```
if (args.length == 0) {
    System.err.println(
        "Usage: java AttributeClient protocol://username@host/foldername");
    return;
}

URLConnection server = new URLName(args[0]);

try {

    Session session = Session.getDefaultInstance(new Properties(),
        new MailAuthenticator(server.getUsername()));

    // CVerbindungsaufbau, Folder öffnen
    Folder folder = session.getFolder(server);
    if (folder == null) {
        System.out.println("Folder " + server.getFile() + " nicht
gefunden.");
        System.exit(1);
    }
    folder.open(Folder.READ_ONLY);

    // Get the messages from the server
    Message[] messages = folder.getMessages();
    for (int i = 0; i < messages.length; i++) {
        System.out.println("----- Message " + (i+1)
            + " -----");
        String from = InternetAddress.toString(messages[i].getFrom());
        if (from != null) System.out.println("From: " + from);
        String to = InternetAddress.toString(
            messages[i].getRecipients(Message.RecipientType.TO));
        if (to != null) System.out.println("To: " + to);
        String subject = messages[i].getSubject();
        if (subject != null) System.out.println("Subject: " + subject);
        Date sent = messages[i].getSentDate();
        if (sent != null) System.out.println("Sent: " + sent);

        System.out.println();
        // Attribute
        System.out.println("Die Nachricht ist ungefähr "
            + messages[i].getSize() + " Bytes lang.");
        System.out.println("Die Nachricht umfasst ungefähr "
            + messages[i].getLineCount() + " Zeilen.");
        String disposition = messages[i].getDisposition();
        if (disposition == null) ; // nichts
        else if (disposition.equals(Part.INLINE)) {
            System.out.println("Inline Anzeige");
        }
        else if (disposition.equals(Part.ATTACHMENT)) {
            System.out.println("Attachment");
            String fileName = messages[i].getFileName();
            if (fileName != null) {
                System.out.println("Dateinamen des Attachments "
                    + fileName);
            }
        }
        String description = messages[i].getDescription();
        if (description != null) {
            System.out.println("Description der Nachricht "
                + description);
        }
    }
}
```

# NETZWERKPROGRAMMIERUNG IN JAVA

```
    }
    // Verbindungsabbau
    // Meldungen bleiben auf dem Server
    folder.close(false);
}
catch (Exception e) {
    e.printStackTrace();
}
// GUI abschliessen
// exit()
System.exit(0);
}
}
```

----- Message 1 -----

From:

Sandra Berger - Sun Switzerland Zurich - Educational Telesales  
<Sandra.Berger@swiss.sun.com>

To: jjoller@hta.fhz.ch

Subject: Zertifizierungen: Schulungen von Sun Educational Services

Sent: Mon Dec 11 11:49:27 GMT+01:00 2000

Die Nachricht ist ungefähr 1648 Bytes lang.

Die Nachricht umfasst ungefähr -1 Zeilen.

----- Message 2 -----

From: Adrian Jetzer <wwwaci@tic.ch>

To: Ralph Rickenbach <ralph.rickenbach@artificial-life.com>

Subject: Sitzung 21.12.00

Sent: Tue Dec 12 11:34:33 GMT+01:00 2000

Die Nachricht ist ungefähr 2832 Bytes lang.

Die Nachricht umfasst ungefähr -1 Zeilen.

----- Message 3 -----

From: Joller\_Voss Uster <efx3874@bluewin.ch>

To: fam\_joller@bluewin.ch

Subject: Faxabsender: Joller\_Voss Uster<efx3874@bluewin.ch>

Sent: Tue Dec 12 12:24:26 GMT+01:00 2000

Die Nachricht ist ungefähr 23364 Bytes lang.

Die Nachricht umfasst ungefähr -1 Zeilen.

POP3 Server sind nicht besonders interessant!

## 15.11.2. Headers

Klassen, welche das `Part` Interface implementieren, beispielweise `Message`, implementieren in der Regel Methoden, mit denen `To:`, ... gesetzt und gelesen werden können. `Part` seinerseits beschränkt sich auf allgemeinere Aussagen.

```
public String [ ] getHeader(String name) throws MessagingException
public void setHeader(String name, String value) throws .....
public void addHeader(String name) throws .....
public void removeHeader(String name) throws .....
public Enumeration getAllheaders() throws ...
```

Mit der `getHeader(...)` Methode werden jene Nachrichtenheader gelesen, welcher dem Namensargument entspricht.

`setHeader(...)` überschreibt einen Header;

`addHeader(...)` fügt einen weiteren Header hinzu, überschreibt also nicht;

# NETZWERKPROGRAMMIERUNG IN JAVA

`removeHeader(...)` entfernt einen bestimmten oder mehrere Header (match Code)  
`getAllHeaders(...)` liefert alle Header in der nachricht

`public Enumeration getMatchingHeaders(String[] names) throws ...`  
`public Enumeration getNonMatchingHeaders(String[] names) throws ...`  
`getMatching(...)` liefert alle Headers, welche matchen, mit mehreren Strings!

## 15.11.2.1. Header Klasse

Diese Klasse ist sehr einfach und verfügt auch über sehr wenig Methoden:

```
public Header(String name, String value)
public String getName()
public String getValue()
```

### Beispiel 32 Mail Headers

```
package lesenallermailheader;

/**
 * Title:          Lesen aller Header
 * Description:
 * Copyright:      Copyright (c) 2000
 * Company:        Joller-Voss GmbH
 * @author J.M.Joller
 * @version 1.0
 */
import javax.mail.*;
import javax.mail.internet.*;
import java.util.*;

public class AllHeaderClient {

    public static void main(String[] args) {

        if (args.length == 0) {
            System.err.println(
                "Usage: java AllHeaderClient
protocol://username@host:port/foldername");
            return;
        }

        URLName server = new URLName(args[0]);

        try {

            Session session = Session.getDefaultInstance(new Properties(),
                new MailAuthenticator(server.getUsername()));

            // Verbindungsaufbau
            Folder folder = session.getFolder(server);
            if (folder == null) {
                System.out.println("Folder " + server.getFile() + " nicht
gefunden.");
                System.exit(1);
            }
            folder.open(Folder.READ_ONLY);

            // Lesen der Nachrichten
```

# NETZWERKPROGRAMMIERUNG IN JAVA

```
Message[] messages = folder.getMessages();
for (int i = 0; i < messages.length; i++) {
    System.out.println("----- Message " + (i+1)
        + " -----");
    // Unterschied
    Enumeration headers = messages[i].getAllHeaders();
    while (headers.hasMoreElements()) {
        Header h = (Header) headers.nextElement();
        System.out.println(h.getName() + ": " + h.getValue());
    }
    System.out.println();
}

// Verbindungsabbau
// Nachrichten bleiben auf dem Server
folder.close(false);

}

catch (Exception e) {
    e.printStackTrace();
}

// GUI herunterfahren
// exit
System.exit(0);
}
}
```

wobei Sie wie immer daran denken müssen, dass der Parameter nicht einfach die Mailadresse ist! Der Mail Server muss spezifiziert werden!

```
----- Message 1 -----
Return-Path: <JJoller@hta.fhz.ch>
Received: from mta1n.bluewin.ch (172.21.1.230) by mss1n.bluewin.ch (Bluewin AG MSS engine 5.1.053)
    id 3A1BAB8700ABFF29 for fam_joller@bluewin.ch; Mon, 11 Dec 2000 11:48:35 +0100
Received: from ntindex.ztl.ch (147.88.1.197) by mta1n.bluewin.ch (Bluewin AG MX engine 5.1.053)
    id 3A2F9F9D001AB178 for fam_joller@bluewin.ch; Mon, 11 Dec 2000 11:49:35 +0100
Received: by ztnt05.ztl.ch with Internet Mail Service (5.5.2650.21)
    id <XQPC4BDC>; Mon, 11 Dec 2000 11:49:37 +0100
Received: from mercury.Sun.COM ([192.9.25.1]) by ntindex.ztl.ch with SMTP (Microsoft Exchange Internet
Mail Service Version 5.5.2650.21)
    id XQPC4BDB; Mon, 11 Dec 2000 11:49:33 +0100
Received: from ezrh04.Swiss.Sun.COM ([129.156.51.2])
    by mercury.Sun.COM (8.9.3+Sun/8.9.3) with ESMTP id CAA01389
    for <jjoller@hta.fhz.ch>; Mon, 11 Dec 2000 02:49:29 -0800 (PST)
Received: from giornico (giornico [129.156.52.94])
    by ezrh04.Swiss.Sun.COM (8.8.8+Sun/8.8.8/ENSMail,v2.0) with SMTP id LAA12887;
    Mon, 11 Dec 2000 11:49:27 +0100 (MET)
Message-Id: <200012111049.LAA12887@ezrh04.Swiss.Sun.COM>
Date: Mon, 11 Dec 2000 11:49:27 +0100 (MET)
From: Sandra Berger - Sun Switzerland Zurich - Educational Telesales <Sandra.Berger@swiss.sun.com>
Reply-To: Sandra Berger - Sun Switzerland Zurich - Educational Telesales <Sandra.Berger@swiss.sun.com>
Subject: Zertifizierungen: Schulungen von Sun Educational Services
To: jjoller@hta.fhz.ch
Cc: Sandra.Berger@swiss.sun.com
MIME-Version: 1.0
Content-Type: TEXT/plain; charset=ISO-8859-1
Content-Transfer-Encoding: QUOTED-PRINTABLE
Content-MD5: uzzRr8jNNMhzhg19w0yW/g==
X-Mailer: dtmail 1.3.0 @(#)CDE Version 1.3.2 SunOS 5.7 sun4u sparc
```

# NETZWERKPROGRAMMIERUNG IN JAVA

----- Message 2 -----

Return-Path: <wwwaci@tic.ch>

Received: from mta1n.bluewin.ch (172.21.1.230) by mss1n.bluewin.ch (Bluewin AG MSS engine 5.1.053) id 3A1BAB8700B1E3B4 for fam\_joller@bluewin.ch; Tue, 12 Dec 2000 11:31:34 +0100

Received: from ntindex.ztl.ch (147.88.1.197) by mta1n.bluewin.ch (Bluewin AG MX engine 5.1.053) id 3A2F9F9D002555FD for fam\_joller@bluewin.ch; Tue, 12 Dec 2000 11:32:37 +0100

Received: from ztex37 (ztex37.ztl.ch [147.88.74.161]) by ntindex.ztl.ch with SMTP (Microsoft Exchange Internet Mail Service Version 5.5.2650.21) id XQPC41ZZ; Tue, 12 Dec 2000 11:32:38 +0100

Message-ID: <004701c06427\$1e5df4b0\$a14a5893@ztex37>

From: "Adrian Jetzer" <wwwaci@tic.ch>

To: "Ralph Rickenbach" <ralph.rickenbach@artificial-life.com>

Cc: "Josef M. Joller" <fam\_joller@bluewin.ch>

Subject: Sitzung 21.12.00

Date: Tue, 12 Dec 2000 11:34:33 +0100

MIME-Version: 1.0

Content-Type: multipart/alternative;

boundary="-----\_NextPart\_000\_0044\_01C0642F.7FFF4450"

X-Priority: 3

X-MSMail-Priority: Normal

X-Mailer: Microsoft Outlook Express 5.00.2919.6700

X-MimeOLE: Produced By Microsoft MimeOLE V5.00.2919.6700

## 15.11.3. Content

Der Inhalt der Nachricht wird als Bytesequenz dargestellt. Im Falle einer zusammengesetzten Nachricht, kann jeder Part weitere Parts enthalten. Interessant sind auch Attachments wie WAV Dateien oder Videoclips.

### 15.11.3.1. Lesen des Inhalts der Part

Sie können mit zwei Methoden arbeiten, um den Typus einer Nachricht zu bestimmen:

```
public String getContentType() throws MessagingException
public boolean isMimeType() throws MessagingException
```

Die erste Methode liefert beispielsweise text/plain, die zweite prüft, ob der Mime Type übereinstimmt.

### 15.11.3.2. Setzen des Inhalts der Part

Beim Senden muss der Text einer Meldung gesetzt werden. Dazu stehen wie oben allgemeine Methoden zur Verfügung:

```
public void setText(String text) throws MessagingException, IllegalWrite,...
public void setContent(Object o, String type) throws ...
public void setContent(Multipart mp) throws ...
```

Mit der setText Methode wird der Mime Type auf text/plain gesetzt; die Methode setContent(...) setzt den Mime Type auf multipart/mixed oder ähnlich.

Damit haben wir einen Kurzüberblick über die Methoden von Part gewonnen. Wie bereits am Anfang dieses Unterkapitels erwähnt, beschreiben diese Methoden allgemein gültige Teile. In der definierten Unterklassen können und müssen diese überschrieben werden.

# NETZWERKPROGRAMMIERUNG IN JAVA

Es handelt sich also bewusst um allgemeine Methoden. Sonst würde man sicher unter `setContent(...)` eine präzisere, andere Funktion erwarten.

## 15.12. *Multipart Messages und Datei Attachments*

Die Kodierung und Dekodierung von beliebigen Datentypen in 7-bit email Text ist recht clever gelöst und in RFCs beschrieben. JavaMail versteckt alle mühsamen Teile und liefert Methoden, die Details verstecken.

Alles was Sie zu tun haben, ist ein `MimeMultipart` Objekt an die Methode `setContent()` zu übergeben. Die meisten Methoden, um Multipart Messages zu bauen, sind in der abstrakten Klasse `javax.mail.Multipart` enthalten:

```
public abstract class Multipart extends Object
```

In der Regel startet man aber mit einer Implementation beispielsweise

```
public class MimeMultipart extends Multipart
```

Jeden Part, den man Multipart hinzufügt, ist eine Instanz der abstrakten Klasse `javax.mail.BodyPart` :

```
public abstract class BodyPart extends Object implements Part
```

Im InternetMail werden wir konkret eine Unterklasse von `BodyPart` verwenden:

```
public class MimeBodyPart extends BodyPart implements MimePart
```

Die meisten Methoden, die man dafür einsetzt, stammen aus der Klasse `Part`. Wir haben diese also bereits besprochen.

```
public String      getContentType()
public int         getCount() throws MessagingException
public BodyPart   getBodyPart(int index) throws IndexOutOfBoundsException, .
```

Die `getContentType()` Methode liefert den MIME Type., also typischerweise etwas wie `multipart/mixed`, im Gegensatz zu einfachen MIME Types wie `image/gif`.

Die `getCount()` Methode liefert die Anzahl Teile in Multipart.

Die `getBodyPart(...)` liefert einen bestimmten Part.

Schauen wir uns ein Beispiel an.

### **Beispiel 33 Multipart Message mit angehängten Dateien**

```
package multipartnachrichtmitdateien;
```

```
/**
 * Title:      Multipart Message mit mehreren angehängten Dateien
 * Description: Multipart MIME
 * Copyright:   Copyright (c) 2000
 * Company:    Joller-Voss GmbH
 * @author J.M.Joller
 * @version 1.0
 */
```

```
import javax.mail.*;
import javax.mail.internet.*;
```

# NETZWERKPROGRAMMIERUNG IN JAVA

```
import java.util.*;
import java.io.*;

public class AllPartsClient {

    public static void main(String[] args) {

        if (args.length == 0) {
            System.err.println(
                "Usage: java SearchClient
protocol://username@host:port/foldername");
            return;
        }

        URLName server = new URLName(args[0]);

        try {

            Session session = Session.getDefaultInstance(new Properties(),
                new MailAuthenticator(server.getUsername()));

            // Verbindungsaufbau
            Folder folder = session.getFolder(server);
            if (folder == null) {
                System.out.println("Folder " + server.getFile() + " nicht
gefunden.");
                System.exit(1);
            }
            folder.open(Folder.READ_ONLY);

            // Get the messages from the server
            Message[] messages = folder.getMessages();
            for (int i = 0; i < messages.length; i++) {
                System.out.println("----- Message " + (i+1)
                    + " -----");

                // Print message headers
                Enumeration headers = messages[i].getAllHeaders();
                while (headers.hasMoreElements()) {
                    Header h = (Header) headers.nextElement();
                    System.out.println(h.getName() + ": " + h.getValue());
                }
                System.out.println();

                // Enumerate Parts
                Object body = messages[i].getContent();
                if (body instanceof Multipart) {
                    processMultipart((Multipart) body);
                }
                else { // einfache Meldung
                    processPart(messages[i]);
                }

                System.out.println();
            }

            // Verbindungsabbau
            // Meldungen bleiben auf dem Server
            folder.close(false);
        }
    }
}
```

# NETZWERKPROGRAMMIERUNG IN JAVA

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

// GUI herunter fahren
// exit
System.exit(0);
}

public static void processMultipart(Multipart mp)
throws MessagingException {

    for (int i = 0; i < mp.getCount(); i++) {
        processPart(mp.getBodyPart(i));
    }
}

public static void processPart(Part p) {

    try {
        String fileName = p.getFileName();
        String disposition = p.getDisposition();
        String contentType = p.getContentType();
        if (fileName == null && (disposition.equals(Part.ATTACHMENT)
        || !contentType.equals("text/plain"))) {
            // Zufallsnamen
            fileName = File.createTempFile("attachment", ".txt").getName();
        }
        if (fileName == null) { // inline
            p.writeTo(System.out);
        }
        else {
            File f = new File(fileName);
            // suchen einer Version, die noch nicht existiert
            for (int i = 1; f.exists(); i++) {
                String newName = fileName + " " + i;
                f = new File(newName);
            }
            FileOutputStream out = new FileOutputStream(f);

            // p.writeTo() funktioniert nicht
            // Wir kopieren Input auf Output Stream
            // Codierung erfolgt automatisch
            // Base-64, verschiedene Formate
            InputStream in = new BufferedInputStream(p.getInputStream());
            int b;
            while ((b = in.read()) != -1) out.write(b);
            out.flush();
            out.close();
            in.close();
        }
    }
    catch (Exception e) {
        System.err.println(e);
        e.printStackTrace();
    }
}
}
```

# NETZWERKPROGRAMMIERUNG IN JAVA

## 15.13. *MIME Messages*

MIME wurde für das Internet Mail entwickelt. Daher ist in der Regel eine Internet Nachricht eine MIME Nachricht.

```
public class MimeMessage extends Message implements MimePart
```

Diese Klasse verfügt über etwa siebzig (70) `public` und `protected` Methoden. Die meisten davon überschreiben Methoden der Klasse `Message` oder implementieren Methoden von `Part` und `MimePart`.

Beispielweise kann mit einer Methode die `MessageID` bestimmt werden. Auch der Zugriff auf alle andern Datenfelder wird über Methoden sichergestellt.

## 15.14. *Folders*

Bisher haben wir lediglich mit der INBOX gearbeitet. Daneben existieren aber viele weitere Folder, die Sie zum Teil selbst definieren können.

```
public abstract class Folder extends Object
```

Die meisten der dazugehörigen Methoden sind abstrakt und müssen passend implementiert werden. Speziell für den Zugriff auf IMAP, mbox, ... sind solche Mechanismen nötig. Die Implementierung in der Sun Referent Implementation fehlt jedoch. Sie müssten also eine Implementation auf dem Web suchen.

### 15.14.1. Öffnen von Folders

Folders kann man nicht so einfach konstruieren. Den einzigen Konstruktor

```
protected Folder(Store store)
```

Aber Sie erhalten Folders auch von `Session`, `Store` oder andern Folders. Beispielsweise:

```
Folder outbox = container.getFolder("gesendet");
```

Ob ein Folder existiert, kann abgefragt werden:

```
public boolean exists() throws MessagingException
```

Wann immer man einen Folder erhält ist dieser geschlossen und Sie müssen ihn öffnen; sie können auch prüfen ob er offen ist und ihn schliesslich wieder schliessen:

```
public abstract void open(int mode) throws ...
public abstract boolean isOpen()
public abstract void close(boolean expurge) throws ...
```

### 15.14.2. Grundlegende Informationen über Folder

Die Folder Klasse besitzt acht Methoden, welche grundlegende Informatione nliefern:

```
public abstract String getName()
public abstract String getFullName()
public URLName getURLName() throws ...
```

# NETZWERKPROGRAMMIERUNG IN JAVA

```
public abstract Folder getParent() throws ...
public abstract int    getType() throws ...
public int             getMode() ...
public Store           getStore()
public abstract char   getSeparator() throws ...
```

Der Name (get...) ist beispielsweise "KontakteETH"; mit getFull... erhält man die gesamte Hierarchie ("aFuE/Schweiz/KontakteETH"). Die getURL... liefert nach den Server, beispielsweise "imap://josef.m.joller@mail.swissonline.ch/aFuE/Schweiz/KontakteETH".

Bei getParanet... erhält man das übergeordnete Verzeichnis, beispielsweise ../Schweiz/...

## 15.14.3. Verwalten von Folders

Ein neuer Folder wird mit create... kreiert, wie auch sonst:

```
public abstract boolean create(int type) throws MessagingException
```

wobei hier unter TYPE Folder.HOLD\_MESSAGES und FOLDER.HOLDS\_FOLDERS zu verstehen ist.

Das Löschen eines Folders geschieht, überraschenderweise, durch delete...

```
public abstract boolean delete(boolean recurse) throws .....
```

Der Parameter recurse bestimmt, ob auch Unterfolder gelöscht werden. Falls dieser Parameter false ist, es aber Unterfolder gibt, kann nichts ausgeführt werden, wie in UNIX.

Folder können auch umbenannt werden:

```
public abstract boolean renameTo(Folder f) throws .....
```

oder die Nachrichten daraus kopiert werden:

```
public void copyMessages(Message[] messages, Folder destination) throws ...
```

## 15.14.4. Subskription

Offiziell werden auch Subskriptionen unterstützt. Dies würde beispielsweise einem Benutzer erlauben sich bei einer Newsgroup einzutragen. Aber in der Regel wird diese Muster von POP, IMAP und vielen andern Providern nicht implementiert.

```
public boolean    isSubscribed()
public void       setSubscribed(boolean subscribe) throws ...
```

## 15.14.5. Inhalt eines Folders

Folder sind typischerweise hierarchisch. Folder können also andere Folder enthalten. Es gibt vier Methoden, um den Inhalt von Folders aufzulisten:

```
public Folder[] list() throws ...
public Folder[] listSubscribed() throws ...
public abstract Folder[] list(String pattern) throws ...
public Folder[] listSubscribed(String pattern) throws ...
```

# NETZWERKPROGRAMMIERUNG IN JAVA

Patterns können auch \* oder den Platzhalter % enthalten, analog zu Verzeichnislisten in DOS, UNIX, ...

## 15.14.6. Prüfen ob Mails vorhanden sind

Methoden:

```
public abstract int getMessageCount() throws...
public abstract boolean hasNewMessages() throws ...
public int getNewMessageCount() throws ...
public int getUnreadMessageCount() throws ...
```

Falls eine der Methoden die Anzahl Nachrichten nicht bestimmen kann, wird -1 zurück gegeben (unbestimmt).

## 15.14.7. Messages aus Foldern

Aus offenen Foldern können Nachrichten gelesen werden:

```
public abstract Message getMessage(int messageNumber) throws ...
public Message[ ] getMessage() throws ...
public Message[ ] getMessage(int start, int end) throws ...
public Message[ ] getMessages(int[ ] messageNumber) throws ...
```

Die Interpretation dieser Methoden folgt aus der Erklärung der weiter vorne gegebenen.

## 15.14.8. Durchsuchen von Foldern

Fie meisten IMAP plus einige POP3 Server erlauben auch eine Suche, ein Durchsuchen der Nachrichten nach bestimmten Kriterien.

### Beispiel 34 Durchsuchen aller Mails nach Sender (From: )

```
package durchsuchenvonfolders;

/**
 * Title:          Durchsuchen von Foldern nach From
 * Description:
 * Copyright:      Copyright (c) 2000
 * Company:        Joller-Voss GmbH
 * @author J.M.Joller
 * @version 1.0
 */
import javax.mail.*;
import javax.mail.search.*;
import javax.mail.internet.*;
import java.util.*;
import java.io.*;

public class SearchClient {

    public static void main(String[] args) {

        if (args.length == 0) {
            System.err.println(
                "Usage: java AllPartsClient
protocol://username@host:port/foldername");
            return;
        }
    }
}
```

# NETZWERKPROGRAMMIERUNG IN JAVA

```
}

URLConnection server = new URLConnection(args[0]);

try {

    Session session = Session.getDefaultInstance(new Properties(),
        new MailAuthenticator(server.getUsername()));

    // Verbindungsaufbau
    Folder folder = session.getFolder(server);
    if (folder == null) {
        System.out.println("Folder " + server.getFile() + " nicht
gefunden.");
        System.exit(1);
    }
    folder.open(Folder.READ_ONLY);

    SearchTerm term = null;
    if (args.length > 1) {
        SearchTerm[] terms = new SearchTerm[args.length-1];
        for (int i = 1; i < args.length; i++) {
            Address a = new InternetAddress(args[i]);
            terms[i-1] = new FromTerm(new InternetAddress(args[i]));
        }
        if (terms.length > 1) term = new OrTerm(terms);
        else term = terms[0];
    }

    // Nachricht lesen
    Message[] messages;
    if (term == null) {
        messages = folder.getMessages();
    }
    else {
        messages = folder.search(term);
    }
    for (int i = 0; i < messages.length; i++) {
        System.out.println("----- Message " + (i+1)
            + " -----");

        // Headers ausgeben
        Enumeration headers = messages[i].getAllHeaders();
        while (headers.hasMoreElements()) {
            Header h = (Header) headers.nextElement();
            System.out.println(h.getName() + ": " + h.getValue());
        }
        System.out.println();

        // Parts auflisten
        Object body = messages[i].getContent();
        if (body instanceof Multipart) {
            processMultipart((Multipart) body);
        }
        else { // Standardmeldung
            processPart(messages[i]);
        }

        System.out.println();
    }
}
```

# NETZWERKPROGRAMMIERUNG IN JAVA

```
// Close the connection
// but don't remove the messages from the server
folder.close(false);

}
catch (Exception e) {
    e.printStackTrace();
}

// Since we may have brought up a GUI to authenticate,
// we can't rely on returning from main() to exit
System.exit(0);

}

public static void processMultipart(Multipart mp)
    throws MessagingException {

    for (int i = 0; i < mp.getCount(); i++) {
        processPart(mp.getBodyPart(i));
    }

}

public static void processPart(Part p) {

    try {
        // I'd prefer to test the Content-Disposition header here.
        // However, too many common email clients don't use it.
        String fileName = p.getFileName();
        if (fileName == null) { // likely inline
            p.writeTo(System.out);
        }
        else if (fileName != null) {
            File f = new File(fileName);
            // find a version that does not yet exist
            for (int i = 1; f.exists(); i++) {
                String newName = fileName + " " + i;
                f = new File(newName);
            }
            FileOutputStream out = new FileOutputStream(f);

            // We can't just use p.writeTo() here because it doesn't
            // decode the attachment. Instead we copy the input stream
            // onto the output stream which does automatically decode
            // Base-64, quoted printable, and a variety of other formats.
            InputStream in = new BufferedInputStream(p.getInputStream());
            int b;
            while ((b = in.read()) != -1) out.write(b);
            out.flush();
            out.close();
            in.close();
        }
    }
    catch (Exception e) {
        System.err.println(e);
        e.printStackTrace();
    }
}
}
```

## **15.15. Schlussbemerkungen**

Damit schliessen wir den klassischen Teil der Netzwerkprogrammierung eigentlich ab. JavaMail ist Teil der Java Enterprise Edition, mit Enterprise Java Beans, RMI (remote methode invocation), IIOP (Internet Inter Orb Protocol), JMS (Java Messaging Services), Servlets und vielemehr, beispielsweise Java Server Pages.

Damit wir mit Threads besser zurecht kommen werden wir uns nun mit "Concurrent Programming in Java - Design Patterns" beschäftigen und einiges an Theorie über verteilte Systeme und deren Beschreibung in Java lernen.

# NETZWERKPROGRAMMIERUNG IN JAVA

<b>SMTP, POP3 UND JAVAMAIL</b> .....	<b>1</b>
15.1. UM WAS GEHT'S EIGENTLICH?.....	1
15.2. SMTP (SIMPLE MESSAGE TRANSFER PROTOCOL) GEMÄSS RFC 821 .....	2
15.3. POP3 (POST OFFICE PROTOCOL V3) GEMÄSS RFC 1725 .....	3
15.4. WAS IST DAS JAVA MAIL API?.....	4
15.5. SENDEN VON EMAILS.....	7
15.5.1. <i>Senden einer Meldung aus einer Applikation</i> .....	10
15.5.2. <i>Senden von emails aus einem Applet</i> .....	13
15.6. EMPFANGEN VON EMAILS .....	17
15.7. PASSWORT AUTHENTIFIZIERUNG.....	23
15.8. ADRESSEN .....	27
15.8.1. <i>Die Address Klasse</i> .....	27
15.8.2. <i>Die InternetAddress Klasse</i> .....	28
15.8.3. <i>Die NewsAddress Klasse</i> .....	31
15.9. DIE URLNAME KLASSE.....	32
15.9.1. <i>Die Konstruktoren</i> .....	32
15.9.2. <i>Methoden zum Parsen eines URLName</i> .....	32
15.10. DIE MESSAGE KLASSE.....	35
15.10.1. <i>Kreieren von Nachrichten / Meldungen / Messages</i> .....	36
15.10.1.1. <i>Nachrichten beantworten</i> .....	36
15.10.1.2. <i>Nachrichten aus Foldern lesen</i> .....	36
15.10.2. <i>Header Info</i> .....	36
15.10.2.1. <i>Die From Adresse</i> .....	37
15.10.2.2. <i>Die ReplyTo Adresse</i> .....	37
15.10.2.3. <i>Die Recipient Adresse</i> .....	38
15.10.2.4. <i>Das Subject einer Nachricht</i> .....	38
15.10.2.5. <i>Datum der Nachricht</i> .....	38
15.10.2.6. <i>Speichern der Änderungen</i> .....	41
15.10.3. <i>Flags</i> .....	41
15.10.4. <i>Folders</i> .....	44
15.10.5. <i>Suchen</i> .....	44
15.11. DAS PART INTERFACE.....	44
15.11.1. <i>Attribute</i> .....	45
15.11.2. <i>Headers</i> .....	47
15.11.2.1. <i>Header Klasse</i> .....	48
15.11.3. <i>Content</i> .....	50
15.11.3.1. <i>Lesen des Inhalts der Part</i> .....	50
15.11.3.2. <i>Setzen des Inhalts der Part</i> .....	50
15.12. MULTIPART MESSAGES UND DATEI ATTACHMENTS .....	52
15.13. MIME MESSAGES .....	55
15.14. FOLDERS .....	55
15.14.1. <i>Öffnen von Folders</i> .....	55
15.14.2. <i>Grundlegende Informationen über Folder</i> .....	55
15.14.3. <i>Verwalten von Folders</i> .....	56
15.14.4. <i>Subskription</i> .....	56
15.14.5. <i>Inhalt eines Folders</i> .....	56
15.14.6. <i>Prüfen ob Mails vorhanden sind</i> .....	57
15.14.7. <i>Messages aus Foldern</i> .....	57
15.14.8. <i>Durchsuchen von Foldern</i> .....	57
15.15. SCHLUSSBEMERKUNGEN .....	60