In diesem Kapitel:

JavaMail Theorie

1.1. 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 sehr komplexe email Anwendungen zu entwickeln. Heute gehört JavaMail zur J2EE, zur *Java Enterprise Edition*. Die Referenzimplementation von Sun wird mit Sockets und Streams implementiert und ist scheusslich. 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 (im Prinzip) leicht implementiert werden.

Es ist also sehr leicht, ein email System analog zu Eudora oder Outlook Express in Java 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 SMTP Servers des CODEBASE Servers.

1.2. 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, oder http://www.faqs.org/rfcs/ . Der Einfachheit halber finden Sie diesen RFC auch als PDF bei den Unterlagen.

Der RFC 821 stammt aus dem Jahre 1982 und wurde von Jonathan Postel vorgeschlagen.

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.

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)

```
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 dasselbe mit Hilfe von Sockets realisieren.

Soweit zu SMTP und RFC 821

1.3. 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: .
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 dasselbe mit Hilfe von Sockets realisieren.

Soweit zu POP3 und RFC 1725

1.4. 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 Programmpakt?
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.

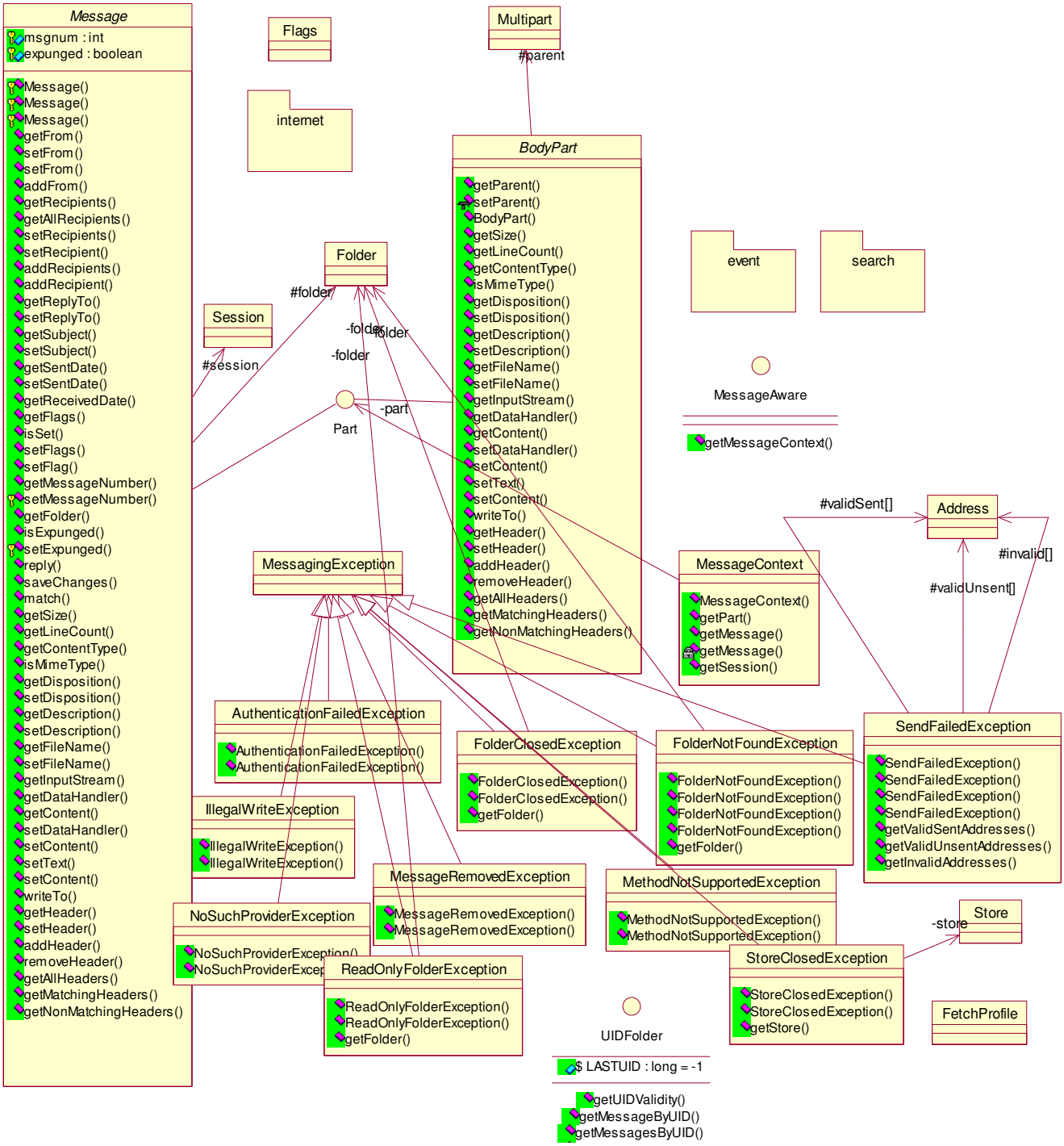
Für das reverse Engineering können Sie eines der gängigen UML Werkzeuge verwenden, beispielsweise Rational Rose oder Together. Die resultierenden Darstellungen dieser beiden Werkzeuge sind etwas unterschiedlich, wegen der nicht so ganz Standard konformen Darstellung im Together.

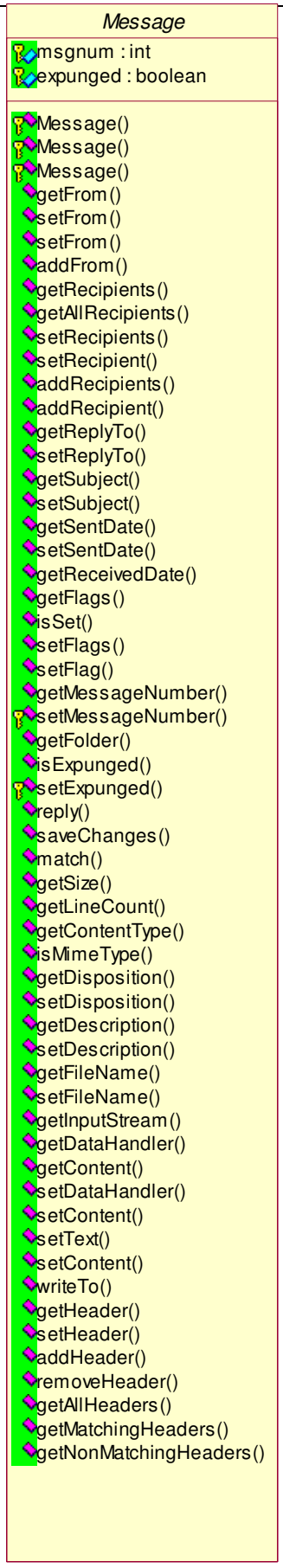
Die Modelle sind sehr umfangreich, typische für reale Anwendungen. Aber das Klassendiagramm, eventuell in einer ersten Version ohne Methoden und Datenfelder, zeigt bereits eine gute Übersicht über die vorhandenen Klassen und Beziehungen untereinander.

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.

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





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 JavaMailAPIs 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, ...) bearbeiten kann,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

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 `jre/lib/ext`. Alternativ dazu kopieren Sie Archive in das \lib Verzeichnis von JDK.

1.5. 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 oder OutlookExpress oder Thunderbird...

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.

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

```
Properties props = new Properties();
props.put("mail.host", "mail.provider.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 from = new InternetAddress("from@host.com", "from.from");
Address to = new InternetAddress("to@ziehlhost.com","to.to");
```

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:oder bcc 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, to);
```

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

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

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.

Beispiel 6 Einfache Text Meldung

```
package theorie;

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", "smtp.swissonline.ch");

      try {
```

```
        // 0) setzen des Mail Hosts (ausgehende Meldungen)
        Session mailConnection = Session.getInstance(props, null);
        mailConnection.setDebug(true);
        // 1) definieren/kreieren:Internet Adressen (Sender/Empfänger)
        Address from = new InternetAddress("from@sender.com", "sender");
        Address to = new InternetAddress("to@empfänger.com");

        Message msg = new MimeMessage(mailConnection);
        // 2) Setzen der From, To und Empfängertypus Felder (TO)
        msg.setFrom(from);
        msg.setRecipient(Message.RecipientType.TO, to);
        msg.setSubject("Besuch?.");
        // 3) setzen des Nachrichtentextes
        msg.setContent("Der Gast aus Moskau spricht schweizerisch...",
         "text/plain");
        // 4) versenden der Nachricht
        Transport.send(msg);
      }
      catch (Exception e) {
        e.printStackTrace();
      }
  }
}
```

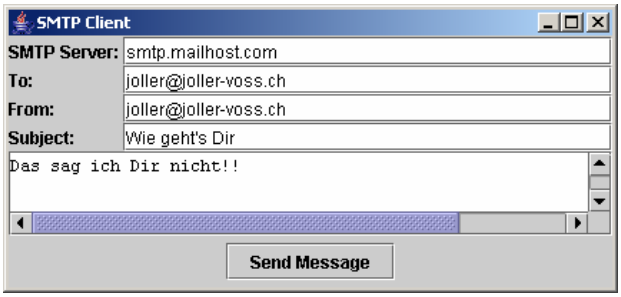
Das Ergebnis ist eine Standardmeldung gemäss RFC 821/822

```
Date: Mon, 8 Mar 2004 21:58:36 +0100
Message-ID: <33520158.1078779510570.JavaMail.from@smtp.sender.com>
From: "sender" <sender@sender.com>
To: to@empfänger.com
Subject: Besuch?.
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
X-wsbox-MailScanner-Information: Please contact the ISP for more
information
X-wsbox-MailScanner: Found to be clean
X-Virus-Scanned: ClamAV version 'clamd / ClamAV version 0.65',
clamav-milter version '0.60p'

Der Gast aus Moskau spricht deutsch, english, schweizerisch...
```

1.5.1. Senden einer Meldung aus einer Applikation

Das obige Beispiel 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



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 theorie;

/**
 * Title:        SMTP Client
 * Aufbau:
 * 0) setzen des Mail Hosts (ausgehende Meldungen)
 * 1) definieren und kreieren Adressen der Sender und Empfänger
 * 2) Setzen der From, To und Empfängertypus Felder (TO)
 * 3) setzen des Nachrichtentextes
 * 4) versenden der Nachricht
 */

import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Properties;

import javax.mail.Address;
import javax.mail.Message;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
import javax.swing.Box;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;

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();
      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());

                        // Zeitverzögern
                        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) {
                        e.printStackTrace();
                }
        }
    }
    public static void main(String[] args) {
        SMTPClient client = new SMTPClient();
        // Java 1.3.
        client.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        client.show();
    }
}
```

Diese Klasse ist alles andere als Spitze:

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

1.5.2. Senden von emails aus einem Applet

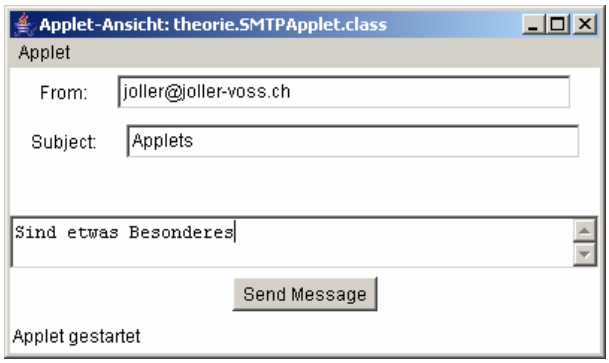
Vom Standpunkt eines GUIs betrachtet, bestehen kaum Unterschiede zwischen einer Anwendung und einem Applet. Allerdings hat der Security Manager des Browsers bei Applets in der Regel etwas gegen zu viel freien Zugriff. Ein Applet kann also lediglich mit dem Host kommunizieren, von dem es stammt.

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="to@empfang.com">
      <PARAM NAME="subject"   VALUE="Was soll das?">
      <PARAM NAME="from"      VALUE=from@sender.com>
</APPLET>
```

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



Achtung: Das Applet verwendet den Start Server als Mailhost:

```
Properties props = new
Properties();

props.put("mail.host",
getCodeBase().getHost());
```

Abbildung 3 SMTPApplet

Beispiel 9 SMTPApplet

```
package theorie;

/**
 * Title:        SMTP Applet
 * Description:  Das Applet liest die Werte der Parameter
 * From:
 * To:
 * und Subject:
 * Diese spezifizieren die entsprechenden Werte für das SMTPApplet
 */

import java.applet.Applet;
import java.awt.BorderLayout;
```

```
import java.awt.Button;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.GridLayout;
import java.awt.Label;
import java.awt.Panel;
import java.awt.TextArea;
import java.awt.TextField;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Properties;

import javax.mail.Address;
import javax.mail.Message;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;

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);

            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());

                        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();
                  }
            }
      }
}
```

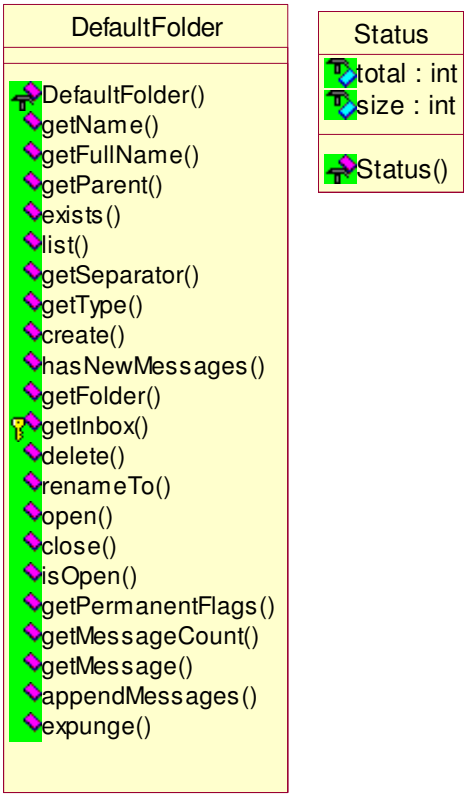
Das Applet kann nicht funktionieren, falls die Archive nicht herunter geladen werden können. Diese müssten also lokal vorhanden sein.

1.6. 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



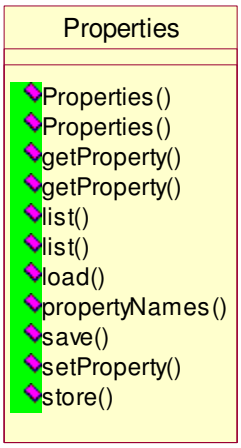
Das JavaMail API geht davon aus, dass der Server (IMAP, NNTP, ...) sowohl Header Informationen als auch die eigentlichen Meldungen verwalten kann. Der Client kann also entweder die Header Information oder die Meldung selbst abfragen. Auch Suchen auf dem Server sollte möglich sein. Der Server speichert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

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.

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

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

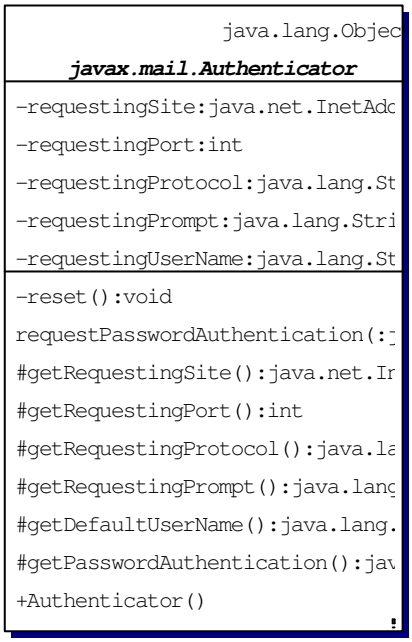


1. `mail.host`
2. `mail.store.protocol`
3. `mail.user`
4. `mail.pop3.user`
5. `mail.pop3.host`

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();
```



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;
```

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);
```

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 **sechsten Schritt** 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.

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 **achte Schritt**: wir brauchen keine weiteren Folder zu öffnen.

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 beim **zehnten Schritt** 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, **der zwölfte Schritt**, 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

```
package theorie;

/** * Title:        Einfacher POP3 Client*/

import java.io.InputStream;
import java.util.Date;
import java.util.Properties;

import javax.mail.Address;
import javax.mail.Folder;
import javax.mail.Message;
import javax.mail.Session;
import javax.mail.Store;
```

```
public class POP3Client {

	public static void main(String[] args) {

		Properties props = new Properties();

		String host = "mail.server.ch";
		String username = "joller";
		String password = "*********";
		String provider = "pop3";

		try {

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

			// Folder öffnen
			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);
			}
			// Verbindungsabbau
			// Nachrichten bleiben auf dem Server
			inbox.close(false);
			store.close();
		} catch (Exception e) {
			e.printStackTrace();
		}
	}
	public static void printMessage(Message m) throws Exception {
		// Header ausdrucken
		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);
		}

		Date d = m.getSentDate();
```

```
            if (d != null) {
                  System.out.println("Date: " + d);
            }
            // Leerzeile als Trennung
            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 {
                  //  multi-part MIME
                  //  führt zu einem Fehler
                  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

```
Return-Path: <sender@sender.com>
Received: from joller-voss.ch (reo0022.wsbox.ch [193.27.218.22])
      by mx.hispeed.ch (8.12.6/8.12.6/tornado-1.0) with ESMTP id
i28LdKRT009399
      for <empfängerr@empfänger.com>; Mon, 8 Mar 2004 22:39:20 +0100
X-Sending-Host: reo0022.wsbox.ch
X-Sending-IP: 193.27.218.22
…
Subject: Besuch?.
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
X-wsbox-MailScanner-Information: Please contact the ISP for more
information
X-wsbox-MailScanner: Found to be clean
X-Virus-Scanned: ClamAV version 'clamd / ClamAV version 0.65', clamav-
milter version '0.60p'

Der Gast aus Moskau spricht deutsch, english, schweizerisch...
```

1.7. 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
```

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 PassworfAuthentication getPasswordAuthentication()
```

Bemerkung 1 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`.

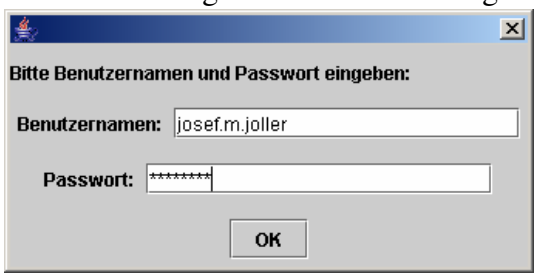
Machen wir uns das Leben besonders einfach und verwenden wir einfach diese Klassen in einem Swing basierten GUI.

Beispiel 22 Eine GUI Authentifizierunsklasse

```
package theorie;
/**
 * Title:        SwingGUImitAuthentifizierung
 * Description:  Einfache Dialogbox
 */
import java.awt.Container;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.mail.Authenticator;
import javax.mail.PasswordAuthentication;
import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JTextField;

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 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.

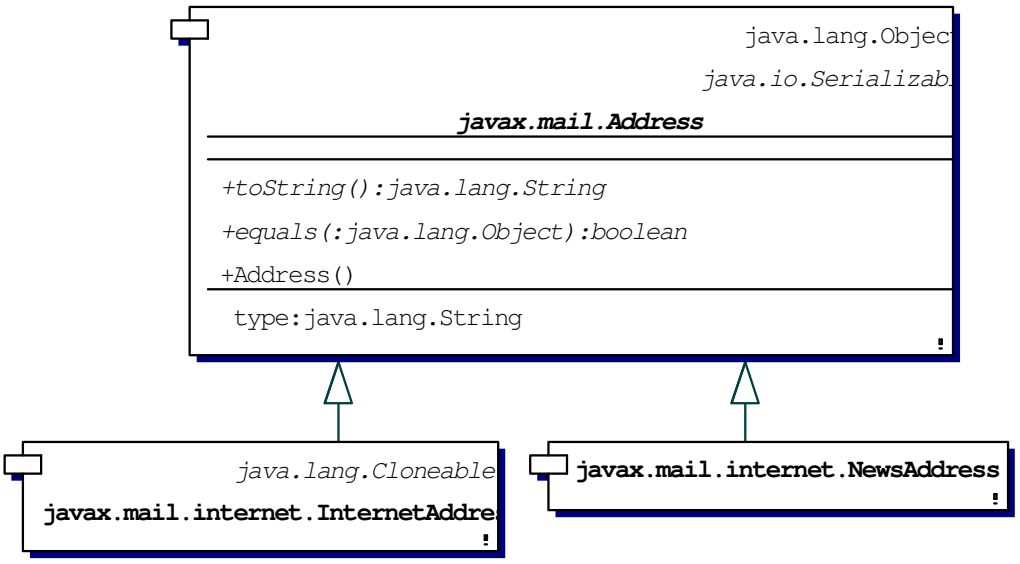
Beispiel 23 Sicherer POP3 Client

```
package theorie;

/**
 * Title:        Sicherer POP3 Client
 * Description:  Ergänzung zum POP3 Client:
 * das Passwort und der Benutzer können zur Laufzeit eingegeben werden.
 */
import java.util.Properties;

import javax.mail.Folder;
import javax.mail.Message;
import javax.mail.Session;
import javax.mail.Store;


public class SichererPOP3Client {
```

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

		Properties props = new Properties();
		String host = "pop3.server.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);
	}
}
```

1.8. Adressen

Schauen wir uns 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 Together) 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 Oberklass, der Wurzel aller Adressklssen.

1.8.1. 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 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`".

1.8.2. 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. joller@joller-voss.ch
2. joller@joller-voss.ch (Josef M. Joller)

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

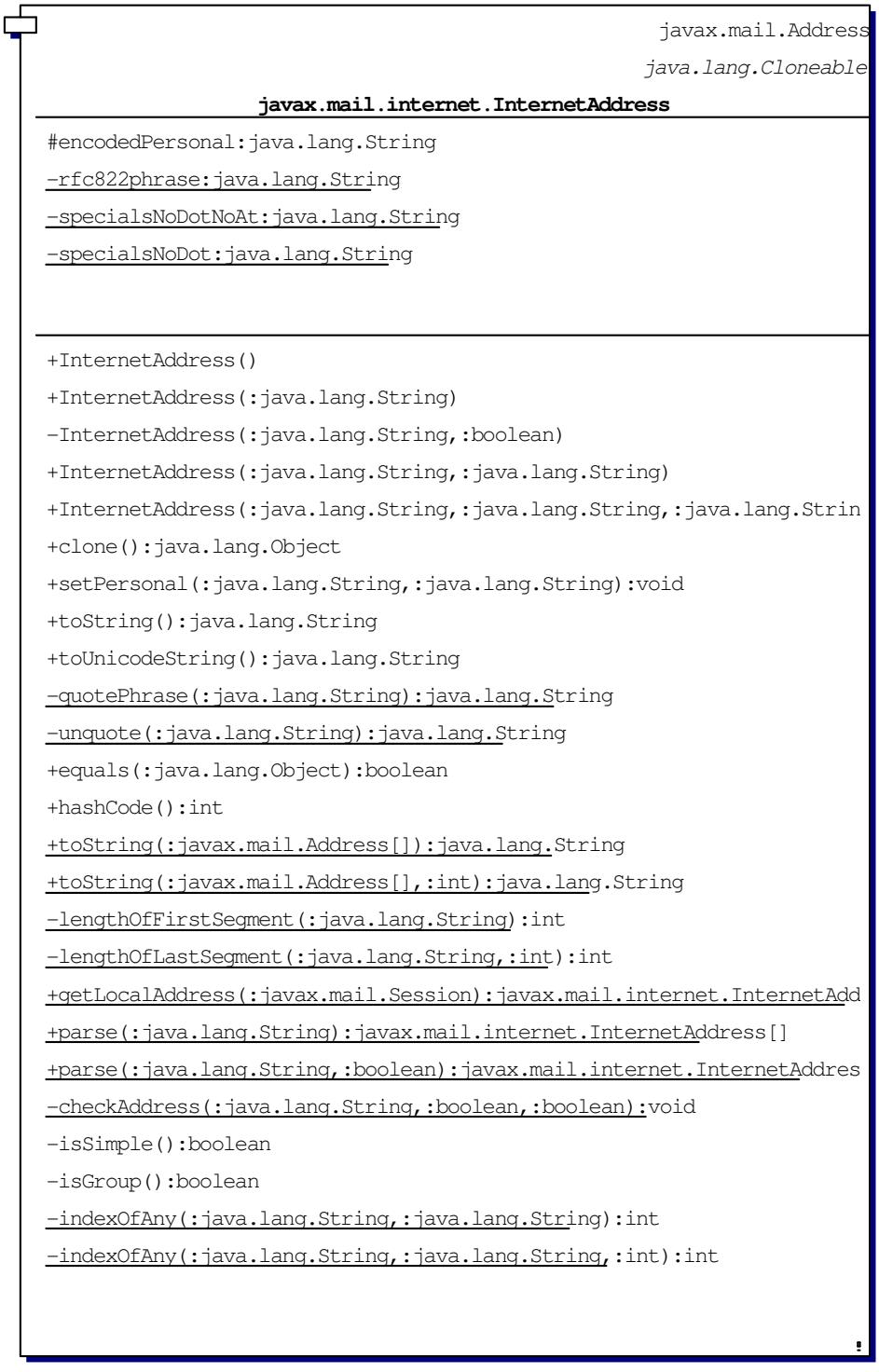
alle Datenfelder sind geschützt und Zeichenketten:

```
protected String
address
personal
encodedPersonal
```

Im obigen Beispiel hätten diese drei Felder folgende Werte:

1. `address` = jjoller@joller-voss.ch
2. `personal` = Josef M. Joller
3. `encodedPersonal`=

Im `encodedPersonal` Datenfeld steht genau dann etwas, wenn beispielsweise ein Name in einer nicht ASCII 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 InternetAddress 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 InternetAddress() {
    }
    public InternetAddress(String address) {
    }
    public InternetAddress(String address, String personal) {
    }
    public InternetAddress(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 InternetAddress("joller@joller-voss.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
    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 InternetAddress[] parse(String addressList) {    }
public static InternetAddress[] 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 `InternetAddress` um

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

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

Die Methode

```
static InternetAddress 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(InternetAddress.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.

1.8.3. Die `NewsAddress` Klasse

Mit einem entsprechenden Provider kann man aus JavaMail auch auf Newsgruppen 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 Newsgruppe und den Host des Objekts. Die `getNewsgroup`() und `getHost`() Methoden liefern die Werte der Newsgruppe und des Host Datenfeldes. Die `getType`() Methode liefert die Zeichenkette "news".

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

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.

1.9. 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.

1.9.1. 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>@mail.server.com/INBOX
Damit können wir auf eine bestimmte INBOX verweisen.

1.9.2. Methoden zum Parsen eines URLName

Diese Methoden sind eigentlich die wichtigsten für diese Klasse:

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

/**
 * Title:        Einfacher MailClient mit URLName als Parameter
 * Description:  Universeller Client, der einen URLName String als
Parameter übernimmt
*/
import java.util.Properties;

import javax.mail.Folder;
import javax.mail.Message;
import javax.mail.Session;
import javax.mail.URLName;

public class ProtokollUnabhaengigerClient {

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

            URLName server = new URLName(args[0]);
            //URLName server = new URLName(temp);
            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: <joller@joller-voss.ch>
Received: from mta1n.bluewin.ch (172.21.1.230) by mss1n.bluewin.ch (Bluewin AG MSS engine 5.1.053)
......
```

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

1.10. 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 vorallem 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, 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`.

1.10.1. 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.

1.10.1.1. 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.

1.10.1.2. 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.

1.10.2. Header Info

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

```
Return-Path: <josef.m.joller@swissonline.ch>
Received: from joller-voss.ch (reo0022.wsbox.ch [193.27.218.22])
      by mx.hispeed.ch (8.12.6/8.12.6/tornado-1.0) with ESMTP id
i28LdKRT009399
      for <josef.m.joller@swissonline.ch>; Mon, 8 Mar 2004 22:39:20 +0100
X-Sending-Host: reo0022.wsbox.ch
X-Sending-IP: 193.27.218.22
Received: from reo0022.wsbox.ch (root@localhost)
      by joller-voss.ch (8.12.10/8.12.10) with ESMTP id i28LddY5013853
      for <joller@joller-voss.ch>; Mon, 8 Mar 2004 22:39:39 +0100
X-ClientAddr: 62.2.95.247
Received: from smtp.hispeed.ch (mxout.hispeed.ch [62.2.95.247])
      by reo0022.wsbox.ch (8.12.10/8.12.10) with ESMTP id i28LdbZI013843
      for <joller@joller-voss.ch>; Mon, 8 Mar 2004 22:39:37 +0100
Received: from NINFPDW11 (217-162-161-78.dclient.hispeed.ch
[217.162.161.78])
      by smtp.hispeed.ch (8.12.6/8.12.6/tornado-1.0) with ESMTP id
i28LdHXE026061
      for <joller@joller-voss.ch>; Mon, 8 Mar 2004 22:39:17 +0100
Date: Mon, 8 Mar 2004 22:39:17 +0100
Message-ID: <33520158.1078781951380.JavaMail.jjoller@smtp.swissonline.ch>
From: "Josef M. Joller" <josef.m.joller@swissonline.ch>
To: joller@joller-voss.ch
Subject: Besuch?.
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
X-wsbox-MailScanner-Information: Please contact the ISP for more
information
X-wsbox-MailScanner: Found to be clean
X-Virus-Scanned: ClamAV version 'clamd / ClamAV version 0.65', clamav-
milter version '0.60p'
Der Gast aus Moskau spricht deutsch, english, schweizerisch...
```

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)

1.10.2.1. 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.

1.10.2.2. 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.

1.10.2.3. 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 formattiert 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.

1.10.2.4. 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.

1.10.2.5. 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 eines einfachen Beispiels, wobei wir lediglich die Headerinformation ausdrucken werden.

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

Beispiel 28 Lesen der Header Informationen

```
package theorie;

/**
 * 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.
 */
import java.util.Date;
import java.util.Properties;

import javax.mail.Folder;
import javax.mail.Message;
import javax.mail.Session;
import javax.mail.URLName;
import javax.mail.internet.InternetAddress;

public class HeaderReaderClient {
```

```
private static final String
            URL="pop3://josef.m.joller@pop.swissonline.ch/inbox";
public static void main(String[] args) {
      String serverURL;
      if (args.length == 0) {
            System.err.println(
                  "Usage: java HeaderClient
                        protocol://username@host:port/foldername");
            //return;
            serverURL=URL;
      } else serverURL=args[0];

      URLName server = new URLName(serverURL);

      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) + " ------------");
                  // Here's the big change...
                  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);

                              System.out.println();
                        }
                        // Schliessen der Verbindung
                        // Meldungen bleiben auf dem Server
                        folder.close(false);
                  } catch (Exception e) {
                        e.printStackTrace();
                  }
                  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: "Josef M. Joller" <josef.m.joller@swissonline.ch>
Reply-to: "Josef M. Joller" <joller@joller-voss.ch>
To: Josef M Joller <josef.m.joller@joller-voss.ch>
Subject: Test Meldung
Sent: Tue Mar 09 18:48:02 CET 2004

------------ Message 2 ------------
From: "Josef M. Joller" <josef.m.joller@swissonline.ch>
Reply-to: "Josef M. Joller" <joller@joller-voss.ch>
To: josef.m.joller@swissonline.ch
Subject: Header Infos
Sent: Tue Mar 09 18:48:31 CET 2004
```

Bemerkung 2 Received Datum

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

1.10.2.6. 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.

1.10.3. 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
```

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 theorie;

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

```
import javax.mail.Flags;
import javax.mail.Folder;
import javax.mail.Message;
import javax.mail.Session;
import javax.mail.URLName;
import javax.mail.internet.InternetAddress;

/**
 * Setzen und lesen von Message Flags
 */

public class FlagSetzenderClient {
      private static final String URL="imap://jjoller@imap.host.com/INBOX";
      public static void main(String[] args) {
            String mail_url;
            if (args.length == 0) {
                  System.err.println(
                        "Usage: java FlagsClient
                              protocol://username@host:port/foldername\n"
                        + "Beispiel:
                        pop3://josef.m.joller@pop.swissonline.ch/INBOX");

                  //System.exit(2);
                  mail_url = URL;
            } else {
                  mail_url = args[0];
            }
            System.out.println("[FlagsClient]Start");
            URLName server = new URLName(mail_url);
            System.out.println("[FlagsClient]URL : " + mail_url);

            try {

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

                  // Verbindungsaufbau zum Server und öffnen der Folder
                  System.out.println("[FlagsClient]Session  getFolder");
                  Folder folder = session.getFolder(server);
                  if (folder == null) {
                        System.out.println(
                              "[FlagsClient]Folder "
                                    + server.getFile()
                                    + " nicht gefunden.");
                        System.exit(1);
                  }
                  System.out.println("[FlagsClient]Session  openFolder");
                  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();
      }

      // Verbindungsabbau
      // Nachrichten bleiben auf dem Server
```

```
                System.out.println("[FlagsClient]Folder  close");
                folder.close(false);
                System.out.println("[FlagsClient]Ende");

            } catch (Exception e) {
                e.printStackTrace();
            }
            System.exit(0); // GUI Ade
        }
}
```

und hier eine mögliche Ausgabe:

```
Usage: java FlagsClient protocol://username@host:port/foldername
Beispiel: pop3://josef.m.joller@pop.swissonline.ch/INBOX
[FlagsClient]Start
[FlagsClient]URL : imap://jjoller@imap.hsr.ch/INBOX
[FlagsClient]Session  getFolder
[FlagsClient]Session  openFolder
…
------------ Message 51 ------------
From: cschelle@hsr.ch
Reply-to: cschelle@hsr.ch
To: jjoller@hsr.ch
Subject: Buch
Sent: Tue Mar 09 16:01:23 CET 2004
Received: Tue Mar 09 16:01:24 CET 2004
Read
...
------------ Message 54 ------------
From: "LUETHY BUECHER  I. User" <i.user@henry.ch>
Reply-to: "LUETHY BUECHER  I. User" <i.User@henry.ch>
To: jjoller@hsr.ch
Subject: Ihre Bestellung
Sent: Tue Mar 09 17:14:00 CET 2004
Received: Tue Mar 09 17:00:13 CET 2004
Answered
Read

[FlagsClient]Folder  close
[FlagsClient]Ende
```

1.10.4. Folders

In der Regel werden Meldungen, die aus demInternet 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)
```

1.10.5. 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
```

1.11. 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:

- 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

1.11.1. 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 theorie;

import java.util.Date;
import java.util.Properties;

import javax.mail.Folder;
import javax.mail.Message;
import javax.mail.Part;
```

```
import javax.mail.Session;
import javax.mail.URLName;
import javax.mail.internet.InternetAddress;

/**
 * Lesen von Mail Attributen
 *
 */
public class AttributLesenderClient {
      private static final String URL = "imap://jjoller@imap.hos.ch/INBOX";
      public static void main(String[] args) {
            System.out.println("[AttributLesenderClient]main()");
            String mail_url;
            if (args.length == 0) {
                  System.err.println(
                        "Usage: java AttributeClient
                              protocol://username@host/foldername");
                  //return;
                  mail_url = URL;
            } else {
                  mail_url = args[0];
            }
            System.out.println("[AttributLesenderClient.main()]Mailserver
                                                URL : "+mail_url);
            URLName server = new URLName(mail_url);
            try {
                  System.out.println("[AttributLesenderClient.main()]
                                                      Session");
                  Session session =
                        Session.getDefaultInstance(
                              new Properties(),
                              new MailAuthenticator(server.getUsername()));
                  // Verbindungsaufbau, Folder öffnen
                  System.out.println("[AttributLesenderClient.main()]
                                                      Folder");
                  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 Messages auf dem Server
                        System.out.println("[AttributLesenderClient.main()]
                                                      Message[]);
                  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("[AttributLesenderClient.main()]
                                        Attribute");
                    System.out.println(
                        "\tDie Nachricht ist ungefähr "
                            + messages[i].getSize()
                            + " Bytes lang.");
                    System.out.println(
                        "\tDie 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("\tInline Anzeige");
                    } else if (disposition.equals(Part.ATTACHMENT)) {
                        System.out.println("\tAttachment");
                        String fileName = messages[i].getFileName();
                        if (fileName != null) {
                            System.out.println(
                                "\tDateinamen des Attachments " +
                                 fileName);
                        }
                    }
                    String description = messages[i].getDescription();
                    if (description != null) {
                        System.out.println(
                            "\tDescription der Nachricht " +
                            description);
                    }
                }
                // Verbindungsabbau
                // Meldungen bleiben auf dem Server
                System.out.println("[AttributLesenderClient.main()]
                                                Folder schliessen");
                folder.close(false);
            } catch (Exception e) {
                e.printStackTrace();
            }
            System.out.println("[AttributLesenderClient.main()]Ende");
            // GUI abschliessen
            // exit()
            System.exit(0);
        }
}
AttributLesenderClient]main()
Usage: java AttributeClient protocol://username@host/foldername
[AttributLesenderClient.main()]Mailserver URL :
imap://jjoller@imap.hsr.ch/INBOX
[AttributLesenderClient.main()]Session
[AttributLesenderClient.main()]Folder
[AttributLesenderClient.main()]Message[]
...
[AttributLesenderClient.main()]Attribute
      Die Nachricht ist ungefähr 1879 Bytes lang.
      Die Nachricht umfasst ungefähr 6 Zeilen.
------- Message 54 ------
```

```
From: "LUETHY BUECHER  I. User" <i.user@henry.ch>
To: jjoller@hsr.ch
Subject: Ihre Bestellung
Sent: Tue Mar 09 17:14:00 CET 2004

[AttributLesenderClient.main()]Attribute
      Die Nachricht ist ungefähr 2757 Bytes lang.
      Die Nachricht umfasst ungefähr 16 Zeilen.
[AttributLesenderClient.main()]Folder schliessen
[AttributLesenderClient.main()]Ende
```

1.11.2. 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;
`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!

1.11.2.1. 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

```
/*
* LesenAllerheaderClient
*/
package theorie;

import java.util.Enumeration;
import java.util.Properties;

import javax.mail.Folder;
import javax.mail.Header;
import javax.mail.Message;
import javax.mail.Session;
import javax.mail.URLName;
```

```
/**
 * Lesen aller Header
 *
 */

public class LesenAllerHeaderClient {

      private static final String URL = "imap://joller@imap.host.ch/inbox";
      public static void main(String[] args) {
            System.out.println("[LesenAllerHeaderClient]main()");
            String mail_url;
            if (args.length == 0) {
                  System.err.println(
                        "Usage: java LesenAlerlHeaderClient
                         protocol://username@host:port/foldername");
                  //return;
                  mail_url = URL;
            } else {
                  mail_url = URL;
            }
            System.out.println("[LesenAllerHeaderClient.main()]Mail URL :"+
                              mail_url);
            URLName server = new URLName(mail_url);
            try {
                  System.out.println("[LesenAllerHeaderClient.main()]
                                                Session eröffnen");
                  Session session =
                        Session.getDefaultInstance(
                              new Properties(),
                              new MailAuthenticator(server.getUsername()));
                  // Verbindungsaufbau
                  System.out.println("[LesenAllerHeaderClient.main()]
                                                   Folder öffnen");
                  Folder folder = session.getFolder(server);
                  if (folder == null) {
                        System.out.println(
                              "\tFolder " + server.getFile() +
                                          " nicht gefunden.");
                        System.exit(1);
                  }
                  folder.open(Folder.READ_ONLY);
                  // Lesen der Nachrichten
                  System.out.println("[LesenAllerHeaderClient.main()]
                                                Messages lesen");
                  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
                  System.out.println("[LesenAllerHeaderClient.main()]
                                    Folder schliessen");
                  folder.close(false);
```

```
            } catch (Exception e) {
                  e.printStackTrace();
            }     // GUI herunterfahren exit
            System.exit(0);
            System.out.println("[LesenAllerHeaderClient.main()]Ende");
      }
}
```

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

Ausgabe:

```
[LesenAllerHeaderClient]main()
Usage: java LesenAlerlHeaderClient protocol://username@host:port/foldername
[LesenAllerHeaderClient.main()]Mail URL : imap://jjoller@imap.hsr.ch/inbox
[LesenAllerHeaderClient.main()]Session eröffnen
[LesenAllerHeaderClient.main()]Folder öffnen
[LesenAllerHeaderClient.main()]Messages lesen
------------ Message 1 ------------
...
------------ Message 54 ------------
Received: from sid00030.hsr.ch ([152.96.22.30]) by sid00031.hsr.ch with
Microsoft SMTPSVC(5.0.2195.6713);
       Tue, 9 Mar 2004 17:00:13 +0100
Received: from hsrmx1.hsr.ch ([152.96.36.50]) by sid00030.hsr.ch with
Microsoft SMTPSVC(5.0.2195.6713);
       Tue, 9 Mar 2004 17:00:13 +0100
Received: from localhost (unknown [127.0.0.1])
      by hsrmx1.hsr.ch (Postfix) with ESMTP id D9B4E1138CD
      for <jjoller@hsr.ch>; Tue,  9 Mar 2004 16:59:49 +0100 (CET)
Received: from hsrmx1.hsr.ch ([127.0.0.1])
 by localhost (hsrmx1 [127.0.0.1]) (amavisd-new, port 10024) with ESMTP
 id 04880-09 for <jjoller@hsr.ch>; Tue,  9 Mar 2004 16:59:48 +0100 (CET)
Received: from obelix.spectraweb.ch (unknown [194.158.229.31])
      by hsrmx1.hsr.ch (Postfix) with ESMTP id 428541138D6
      for <jjoller@hsr.ch>; Tue,  9 Mar 2004 16:59:15 +0100 (CET)
Received: from mail.henry.ch ([195.141.228.18])
      by obelix.spectraweb.ch (8.12.9/8.12.6) with ESMTP id i29FxEkl023530
      for <jjoller@hsr.ch>; Tue, 9 Mar 2004 16:59:14 +0100
Received: from [195.141.228.17] by mail.henry.ch (GMS
8.00.3078/NY6349.00.7a9c2ddc) with SMTP id egsheaaa for jjoller@hsr.ch;
Tue, 9 Mar 2004 17:05:01 +0000
Received: from  ([192.168.50.170]) by fwall; Tue, 09 Mar 2004 16:51:18
+0100 (NFT)
Message-ID: <003d01c405f1$889a7e60$aa32a8c0@solothurn.buchhaus.ch>
From: "LUETHY BUECHER  I. User" <i.user@henry.ch>
To: <jjoller@hsr.ch>
Subject: Ihre Bestellung
Date: Tue, 9 Mar 2004 17:14:00 +0100
MIME-Version: 1.0
Content-Type: text/plain;
      charset="iso-8859-1"
Content-Transfer-Encoding: 8bit
X-Priority: 3
X-MSMail-Priority: Normal
X-Mailer: Microsoft Outlook Express 5.50.4807.1700
X-MimeOLE: Produced By Microsoft MimeOLE V5.50.4807.1700
Return-Path: i.leisibach@henry.ch
X-OriginalArrivalTime: 09 Mar 2004 16:00:13.0850 (UTC)
FILETIME=[9B90D7A0:01C405EF]

[LesenAllerHeaderClient.main()]Folder schliessen
```

1.11.3. 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.

1.11.3.1. 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 MessaingException
```

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

1.11.3.2. 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 0, 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 Kurzüberblich 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.

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

1.12. Multipart Messages und Datei Attachments

Die Kodierung und Dekodierung von 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 Sring      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 theorie;

import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.util.Enumeration;
import java.util.Properties;

import javax.mail.Folder;
import javax.mail.Header;
import javax.mail.Message;
```

```
import javax.mail.MessagingException;
import javax.mail.Multipart;
import javax.mail.Part;
import javax.mail.Session;
import javax.mail.URLName;

/**
 * Multipart Message mit mehreren angehängten Dateien
 * Multipart MIME
 */

public class MultipartMailClient {
      private static final String URL = "imap://jjoller@imap.hsr.ch/INBOX";
      public static void main(String[] args) {
            System.out.println("[MultipartMailClient]main()");
            String mail_url;
            if (args.length == 0) {
                  System.err.println(
                        "Usage: java MultipartMailClient
                        protocol://username@host:port/foldername");
                  //return;
                  mail_url = URL;
            } else {
                  mail_url = args[0];
            }
            System.out.println("[MultipartMailClient.main()]mail server :"+
                                                            mail_url);
            URLName server = new URLName(mail_url);
            try {
                  System.out.println("[MultipartMailClient.main()]
                                          Session eröffnen");
                  Session session =
                        Session.getDefaultInstance(
                              new Properties(),
                              new MailAuthenticator(server.getUsername()));
                  // Verbindungsaufbau
                  System.out.println("[MultipartMailClient.main()]
                                                Folder eröffnen");
                  Folder folder = session.getFolder(server);
                  if (folder == null) {
                        System.out.println(
                              "Folder " + server.getFile() +
                              " nicht gefunden.");
                        System.exit(1);
                  }
                  folder.open(Folder.READ_ONLY);
                  System.out.println("[MultipartMailClient.main()]
                                                Messages holen");
                  Message[] messages = folder.getMessages();
                  for (int i = 0; i < messages.length; i++) {
                        System.out.println(
                              "--- Message " + (i + 1) + " ---");
                        // Ausgabe der 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();
                        // Multiparts
                        Object body = messages[i].getContent();
```

```
                        if (body instanceof Multipart) {
                              processMultipart((Multipart) body);
                        } else { // einfache Meldung
                              processPart(messages[i]);
                        }
                        System.out.println();
                  }
                  System.out.println("[MultipartMailClient.main()]
                                                Session schliessen");
                  // Verbindungsabbau
                  // Meldungen bleiben auf dem Server
                  System.out.println("[MultipartMailClient.main()]
                                                Folder schliessen");
                  folder.close(false);
            } catch (Exception e) {
                  e.printStackTrace();
            }
            // GUI herunter fahren
            // exit
            System.out.println("[MultipartMailClient]main() Ende");
            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();
                }
            }
}
```

Ausgabe

```
[MultipartMailClient]main()
Usage: java MultipartMailClient protocol://username@host:port/foldername
[MultipartMailClient.main()]mail server : imap://jjoller@imap.hst.ch/INBOX
[MultipartMailClient.main()]Session eröffnen
[MultipartMailClient.main()]Folder eröffnen
[MultipartMailClient.main()]Messages holen
--- Message 1 ---
Return-Path: mc@computer.org

--- Message 55 ---
Received: from sid00030.hsr.ch ([152.96.22.30]) by sid00031.hsr.ch with
Microsoft SMTPSVC(5.0.2195.6713);
       Tue, 9 Mar 2004 20:06:11 +0100
Received: from hsrmx1.hsr.ch ([152.96.36.50]) by sid00030.hsr.ch with
Microsoft SMTPSVC(5.0.2195.6713);
       Tue, 9 Mar 2004 20:06:11 +0100
Received: from localhost (unknown [127.0.0.1])
      by hsrmx1.hsr.ch (Postfix) with ESMTP id CDA8A113F25
      for <JJoller@hsr.ch>; Tue,  9 Mar 2004 20:06:03 +0100 (CET)
Received: from hsrmx1.hsr.ch ([127.0.0.1])
 by localhost (hsrmx1 [127.0.0.1]) (amavisd-new, port 10024) with ESMTP
 id 08841-05 for <JJoller@hsr.ch>; Tue,  9 Mar 2004 20:06:02 +0100 (CET)
Received: from smtp.hispeed.ch (unknown [62.2.95.247])
      by hsrmx1.hsr.ch (Postfix) with ESMTP id 92B8C113F23
      for <JJoller@hsr.ch>; Tue,  9 Mar 2004 20:06:02 +0100 (CET)
Received: from NINFPDW11 (217-162-161-78.dclient.hispeed.ch
[217.162.161.78])
      by smtp.hispeed.ch (8.12.6/8.12.6/tornado-1.0) with SMTP id
i29J61pb026768
      for <JJoller@hsr.ch>; Tue, 9 Mar 2004 20:06:01 +0100
Message-ID: <002101c40609$8feaab50$4ea1a2d9@NINFPDW11>
Reply-To: "Josef M. Joller" <joller@joller-voss.ch>
From: "Josef M. Joller" <josef.m.joller@swissonline.ch>
To: "Joller Josef" <JJoller@hsr.ch>
Subject: Mail mit Textdatei attached
Date: Tue, 9 Mar 2004 20:06:00 +0100
Organization: Sonnenbertgstrasse 73, 8610 USTER
MIME-Version: 1.0
Content-Type: multipart/mixed;
      boundary="----=_NextPart_000_001D_01C40611.F157F2A0"
X-Priority: 3
X-MSMail-Priority: Normal
X-Mailer: Microsoft Outlook Express 6.00.2800.1158
X-MimeOLE: Produced By Microsoft MimeOLE V6.00.2800.1165
Return-Path: josef.m.joller@swissonline.ch
X-OriginalArrivalTime: 09 Mar 2004 19:06:11.0831 (UTC)
FILETIME=[963DA070:01C40609]

[MultipartMailClient.main()]Session schliessen
[MultipartMailClient.main()]Folder schliessen
[MultipartMailClient]main() Ende
```

1.13. 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.

1.14. 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.

1.14.1. 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 Foldern. 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 ...
```

1.14.2. 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 ...
```

```
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/...

1.14.3. Verwalten von Folders

Ein neuer Folder wir 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 ...
```

1.14.4. 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 nict implementiert.

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

1.14.5. 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 ...
```

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

1.14.6. 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).

1.14.7. 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.

1.14.8. Durchsuchen von Foldern

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

Beispiel 34 Durchsuchen aller Mails nach Sender (From:)

```
package theorie;

import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.util.Enumeration;
import java.util.Properties;

import javax.mail.Address;
import javax.mail.Folder;
import javax.mail.Header;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.Multipart;
import javax.mail.Part;
import javax.mail.Session;
import javax.mail.URLName;
import javax.mail.internet.InternetAddress;
import javax.mail.search.FromTerm;
import javax.mail.search.OrTerm;
import javax.mail.search.SearchTerm;

/**
 * Durchsuchen von Foldern
 */
```

```
public class SearchClient {
      private static final String URL = "imap://jjoller@imap.hst.ch/INBOX";
      public static void main(String[] args) {
            System.out.println("[SearchClient]main()");
            String mail_url;
            if (args.length == 0) {
                  System.err.println(
                  "Usage: java AllPartsClient
                        protocol://username@host:port/foldername");
                  //return;
                  mail_url = URL;
            } else {
                  mail_url = args[0];
            }
            System.out.println("[SearchClient]main() : mail url : "+
                                                            mail_url);
            URLName server = new URLName(mail_url);
            try {
                  System.out.println("[SearchClient]main() : Session
                                                            eröffnen");
                  Session session =
                        Session.getDefaultInstance(
                              new Properties(),
                              new MailAuthenticator(server.getUsername()));
                  // Verbindungsaufbau
                  System.out.println("[SearchClient]main() : Folder öffnen
                                                      (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);

                  System.out.println("[SearchClient]main() : Search
                                                            definieren");
                  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
                  System.out.println("[SearchClient]main() :
                                                      Messages 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();
                }
                folder.close(false);
                System.out.println("[SearchClient]main() : Folder
                                                        schliessen");
        } catch (Exception e) {
                e.printStackTrace();
        }
        System.out.println("[SearchClient]main() : Ende");
        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();
                if (fileName == null) { // likely inline
                        p.writeTo(System.out);
                } else if (fileName != null) {
                        File f = new File(fileName);
                        for (int i = 1; f.exists(); i++) {
                                String newName = fileName + " " + i;
                                f = new File(newName);
                        }
                        FileOutputStream out = new FileOutputStream(f);
                        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();
            }
        }
```

Ausgabe

```
...

------------ Message 55 ------------
Received: from sid00030.hsr.ch ([152.96.22.30]) by sid00031.hsr.ch with
Microsoft SMTPSVC(5.0.2195.6713);
       Tue, 9 Mar 2004 20:06:11 +0100
Received: from hsrmx1.hsr.ch ([152.96.36.50]) by sid00030.hsr.ch with
Microsoft SMTPSVC(5.0.2195.6713);
       Tue, 9 Mar 2004 20:06:11 +0100
Received: from localhost (unknown [127.0.0.1])
      by hsrmx1.hsr.ch (Postfix) with ESMTP id CDA8A113F25
      for <JJoller@hsr.ch>; Tue,  9 Mar 2004 20:06:03 +0100 (CET)
Received: from hsrmx1.hsr.ch ([127.0.0.1])
 by localhost (hsrmx1 [127.0.0.1]) (amavisd-new, port 10024) with ESMTP
 id 08841-05 for <JJoller@hsr.ch>; Tue,  9 Mar 2004 20:06:02 +0100 (CET)
Received: from smtp.hispeed.ch (unknown [62.2.95.247])
      by hsrmx1.hsr.ch (Postfix) with ESMTP id 92B8C113F23
      for <JJoller@hsr.ch>; Tue,  9 Mar 2004 20:06:02 +0100 (CET)
Received: from NINFPDW11 (217-162-161-78.dclient.hispeed.ch
[217.162.161.78])
      by smtp.hispeed.ch (8.12.6/8.12.6/tornado-1.0) with SMTP id
i29J61pb026768
      for <JJoller@hsr.ch>; Tue, 9 Mar 2004 20:06:01 +0100
Message-ID: <002101c40609$8feaab50$4ea1a2d9@NINFPDW11>
Reply-To: "Josef M. Joller" <joller@joller-voss.ch>
From: "Josef M. Joller" <josef.m.joller@swissonline.ch>
To: "Joller Josef" <JJoller@hsr.ch>
Subject: Mail mit Textdatei attached
Date: Tue, 9 Mar 2004 20:06:00 +0100
Organization: Sonnenbertgstrasse 73, 8610 USTER
MIME-Version: 1.0
Content-Type: multipart/mixed;
      boundary="----=_NextPart_000_001D_01C40611.F157F2A0"
X-Priority: 3
X-MSMail-Priority: Normal
X-Mailer: Microsoft Outlook Express 6.00.2800.1158
X-MimeOLE: Produced By Microsoft MimeOLE V6.00.2800.1165
Return-Path: josef.m.joller@swissonline.ch
X-OriginalArrivalTime: 09 Mar 2004 19:06:11.0831 (UTC)
FILETIME=[963DA070:01C40609]

Content-Type: multipart/alternative;
      boundary="----=_NextPart_001_001E_01C40611.F157F2A0"


------=_NextPart_001_001E_01C40611.F157F2A0
Content-Type: text/plain;
      charset="iso-8859-1"
Content-Transfer-Encoding: quoted-printable

Das war's
------=_NextPart_001_001E_01C40611.F157F2A0
Content-Type: text/html;
      charset="iso-8859-1"
Content-Transfer-Encoding: quoted-printable
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML><HEAD>
<META http-equiv=3DContent-Type content=3D"text/html; =
charset=3Diso-8859-1">
<META content=3D"MSHTML 6.00.2800.1400" name=3DGENERATOR>
<STYLE></STYLE>
</HEAD>
<BODY bgColor=3D#ffffff>
<DIV><FONT face=3DArial size=3D2>Das war's</FONT></DIV></BODY></HTML>

------=_NextPart_001_001E_01C40611.F157F2A0--
```

1.15. Schlussbemerkungen

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 vielem mehr, beispielsweise Java Server Pages.

Mail Clients kann man immer mal brauchen. Die Kombination mit Mailserver Servlets bietet sich direkt an. Sun liefert dazu einige Beispiele.

Viel Spass beim Mailen und Progammieren.