In diesem Kapitel:

JavaMail Praxis

1.1. Generelles

Möchten Sie Mail Funktionen in Ihre Applikationen einbauen? Dann dürfte Ihnen JavaMail dabei helfen. Natürlich könnten Sie auch gleich selber ein email System bauen, mit Sockets und gemäss den relevanten RFCs. Aber JavaMAil nimmt Ihnen vieles ab und ist erst noch ein standardisiertes API, welches auch in anderen Java Applikationen Verwendung finden könnte. IMAP, POP, SMTP, MIME und diese Internet Protokolle, die im Mail Umfeld eine Rolle spielen, sind berücksichtigt worden. Zusätzlich benötigen Sie das JavaBeans™ Activation Framework (JAF).

1.1.1. Konzepte

Nach Abschluss dieses Moduls sollten Sie:

- die grundsätzlichen Internet Mail Protokolle verstehen SMTP, POP3, IMAP und MIME
- die Architektur des JavaMail Frameworks kennen
- das Zusammenspiel zwischen dem JavaMail und dem JavaBeans Activation Framework in etwa kennen.

1.1.2. Lernziele

Nach Abschluss dieses Moduls sollten Sie in der Lage sein:

- Mails mit Hilfe vom JavaMail API zu verschicken
- Mails mit Hilfe vom JavaMail API zu lesen
- Attachments zu senden und zu empfangen
- HTML Messages zu senden und zu empfangen
- Bestimmte Meldungen mit Hilfe von Suchbegriffen zu finden

1.1.3. Voraussetzungen

Sie finden in diesen Kursunterlagen die Hinweise auf die einzusetzende Software. Zudem benötigen Sie JDK Java 2 Standard Edition.

Zudem sollten Sie mit den Grundlagen der Objekt-orientierten Programmierung und Java vertraut sein.

1.1.4. Einführung in das JavaMail API

Das JavaMail™ API ist ein optionales Package (eine Standard Extension) zum Lesen, Erstellen und Versenden von elektronischen Meldungen. Damit können Sie beispielsweise *Mail User Agent* (MUA) Programme analog zu Eudora, Pine und Microsoft Outlook kreieren. Ziel des APIs ist es aber nicht, Messages zu transportieren, abzuliefern oder weiterzuleiten, etwa wie das Unix sendmail oder andere *Mail Transfer Agent* (MTA) Programme.

In andern Worten: der Benutzer arbeitet mit einem MUA ähnlichen Programm, um Messages zu lesen und zu schreiben. Die MUAs hängen von den MTAs ab. Diese liefern die Meldungen ab oder holen diese ab.

JavaMail stellt einen protokollunabhängigen Zugriff auf Messages dar, zum Senden und Empfangen. Das API besteht aus zwei Teilen:

- der erste Teil des APIs wird in diesem Modul behandelt.
  Dabei geht es um das Senden und Empfangen von Messages, unabhängig vom Provider / Protokoll.
- der zweite Teil behandelt protokollabhängige Fragen, wie etwa SMTP, POP, IMAP und NNTP.
  Damit Sie mit JavaMail mit einem Server kommunizieren können, benötigen Sie einen *Provider* für ein Protokoll. Das Schreiben dieser protokollabhängigen Provider wird hier nicht behandelt. Sun bietet aber bereits Provider für die gängigen Protokolle an. Es ist also äusserst unwahrscheinlich, dass Sie je einen eigenen Provider schreiben müssen.

1.2. Übersicht über sie relevanten Protokolle

Bevor wir und die Finessen des JavaMail APIs anschauen, betrachten wir kurz die relevanten Protokolle, welche im Zusammenhang mit dem API eingesetzt werden:

- SMTP
- POP
- IMAP
- MIME

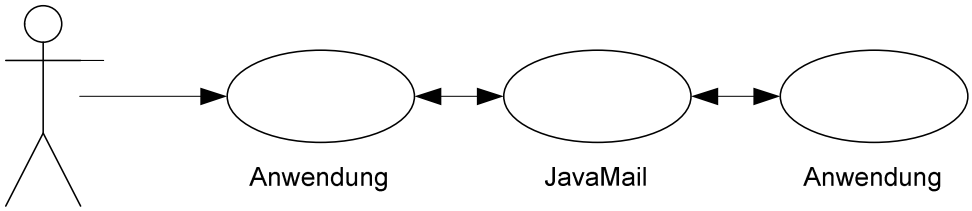
Zusätzlich werden wir noch NNTP kennen lernen. Ein Grundverständnis der Protokolle erleichtert das Verständnis des JavaMail APIs. Obschon das JavaMail API versucht, die Details der Protokolle zu verbergen, erleichtert es das Verständnis, wenn Sie diese kennen.

Falls irgend ein Feature vom Protokoll nicht unterstützt wird, wird es auch in JavaMail kaum auftauchen. Dies kann bereits bei POP der Fall sein, wie Sie gleich sehen werden.

1.2.1. SMTP

Das Simple Mail Transfer Protocol (SMTP) wird im RFC 821 (http://www.faqs.org/rfcs/rfc821.html) definiert. Das Protokoll definiert die grundlegenden Mechanismen zum Abliefern eines emails.

Das JavaMail API bzw. JavaMail basierte Programme werden mit dem Internet Service Provider (ISP) SMTP Server kommunizieren. Dieser wird die Meldung an den Empfänger weiterleiten, in der Regel indirekt über weitere Hosts. Der Empfänger wird vermutlich die Meldung mittels POP oder IMAP lesen.



JavaMail enthält keinerlei Funktionen zum Konfigurieren oder Weiterleiten von Meldungen oder zum Eröffnen oder Löschen von Mail Accounts.

1.2.2. POP

POP steht für Post Office Protocol. Die aktuelle Version ist 3, auch als POP3 bekannt. RFC 1939 (http://www.faqs.org/rfcs/rfc1725.html) definiert dieses Protokoll. POP ist der in der Regel eingesetzte Mechanismus, wie man im Internet Mails empfängt. POP unterstützt eine Mailbox pro Benutzer. Das ist auch schon fast alles was POP liefert. Die Anzahl neuer Meldungen ist beispielsweise keine POP3 Funktion! Diese Funktionalität muss ins Anwenderprogramm eingebaut werden (beispielsweise die Berechnung der Anzahl neuer Meldungen seit Ihrem letzten Besuch Ihrer Mailbox). Daher müssen Sie auch in JavaMail diese Grössen selber berechnen, sofern Sie an diesen Informationen interessiert sind!

1.2.3. IMAP

IMAP ist ein fortschrittlicheres Protokoll zum Empfangen von Meldungen. Es wurde im RFC 2060 definiert (http://www.faqs.org/rfcs/rfc1064.html), IMAP steht für Internet Message Access Protocol, die aktuelle Version ist 4, auch bekannt unter IMAP4. Falls Sie IMAP als Protokoll verwenden wollen, muss vorallem Ihr Mail Server dieses Protokoll unterstützen!

Sie können nicht einfach Ihr Programm auf IMAP umstellen und POP durch IMAP ersetzen. Aber falls der Server IMAP unterstützt, können Sie in JavaMail auch Folder auf dem Server anlegen und diese können von mehreren Benutzern gleichzeitig eingesetzt werden.

Vielleicht denken Sie nun, dass IMAP wegen der höheren Funktionalität universell eingesetzt würde. Falsch!

Der Grund liegt sicher darin, dass IMAP den Server viel stärker belastet. Der Server muss Meldungen empfangen, versenden und aufbewahren, organisieren in mehreren Foldern. Der Vorteil ist beispielsweise, dass mit dem Backup des Servers auch alle Ihre Folder gesichert werden. Aber bei umfangreichen Foldern steigt der Platzbedarf beträchtlich. Beim POP wird die Last schnell an den Client angegeben.

1.2.4. MIME

MIME steht für Multipurpose Internet Mail Extensions. Es handelt sich dabei nmicht um ein Mail Transfer Protokoll. Vielmehr geht es um den Inhalt dessen, was übermittelt wird: das Format der Meldungen, der Anhänge und so weiter. MIME unfasst mehrere RFCs: RFC 822 http://www.imc.org/rfc822 , RFC 2045, RFC 2046 und RFC 2047.In der Regel müssen Sie sich als Benutzer eines Mail Systems keine Gedanken üüber diese RFCs machen. Aber sie existieren und werden von Ihren Mail Programmen verwendet bzw. implementiert.

1.2.5. NNTP und Andere

Weil das JavaMail API zwischen Providern und dem Rest unterscheidet, können Sie leicht neue zusätzliche Protokolle einfügen. Sun unterhält eine Liste der 3rd Party Anbieter (http://java.sun.com/products/javamail/Third_Party.html, welche Anwendungen oder Erweiterungen anbieten.

Dort finden Sie beispielsweise (frei erhältliche) Implementationen von NNTP (Network News Transport Protocol) [newsgroups], S/MIME (Secure Multipurpose Internet Mail Extensions), und weiteren.

1.3. Installation

JavaMail kann man direkt herunterladen oder als Teil der Java Enterprise Edition nutzen.

Die JavaMail Version bei http://java.sun.com/products/javamail/ (Version 1.3.1) lässt sich einfach entzippen und schon stehen die benötigten Bibliotheken und Java Archive zur Verfügung.

Beim Einsatz der Enterprise Edition finden Sie die javax.mail.* Klassen im Java Archiv j2ee.jar.

Nach der Installation von JavaMail müssen Sie das JavaBeans™ Activation Framework installieren: http://java.sun.com/beans/glasgow/jaf.html, siehe unten.

1.3.1. Installation von JavaMail

Um JavaMail 1.1.3 einsetzen zu können, müssen Sie `javamail1.3.1.zip` herunterladen und das Archiv `mail.jar` Ihrem CLASSPATH hinzufügen. In der Version 1.1.3 sind Implementationen von SMTP und IMAP4 Provider enthalten, neben den Core Klassen.

Falls Sie zusätzlich POP benötigen, können Sie von http://java.sun.com/products/javamail/pop3.html einen POP3 Provider herunter laden.
In diesem Fall müssen Sie das Archiv `pop3.jar` Ihrem CLASSPATH hinzufügen.

Nach der Installation von JavaMail müssen Sie das JavaBeans™ Activation Framework installieren.

1.3.2. Installation des JavaBeans Activation Framework

Alle Versionen des JavaMAil API benötigen das JavaBeans Activation Framework. Das Framework unterstützt Datenblöcke und das gesamte Handling von Datenblöcken. Klingt nach wenig, aber dies ist die Grundlage der MIME Implementierung. Das Framework kann man ab http://java.sun.com/products/javabeans/jaf/index.jsp herunterladen. Auch hier muss das Archiv aus `jaf....zip` , `activation.jar` Ihrem CLASSPATH hinzugefügt werden.

Somit haben Sie also im Falle von JavaMail die Archive `mail.jar, eventuell pop3.jar` und `activation.jar` Ihrem CLASSPATH hinzugefügt.

Im Falle von JavaMail 1.2 sind es die Archive `mail.jar, pop3.jar` und `activation.jar` Ihrem CLASSPATH hinzugefügt. Falls Sie POP3 nicht einsetzen wollen, fehlt natürlich dieses Archiv.

Falls Sie Ihren CLASSPATH nicht verändern wollen, können Sie die Archive auch in das Verzeichnis `lib/ext` im Java Runtime Environment (JRE) Verzeichnis verschieben: `%JAVA_HOME%\jre\lib\ext.`

1.3.3. Einsatz mit Java 2 Enterprise Edition

Falls Sie J2EE installiert haben, müssen Sie nichts tun, um die Grundfunktionen von JavaMail nutzen zu können, da diese Java Version bereits mit JavaMail kommt. Einzig das Archiv `j2ee.jar` muss in Ihrem CLASSPATH sein.

Bei J2EE muss der POP3 Provider separat installiert werden. Der Ablauf ist genau gleich wie oben unter JavaMail 1.1.3. Das Activation Framework müssen Sie nicht mehr installieren.

1.4. Mails mit Telnet senden und empfangen

Damit Sie JavaMail oder ein anderes Mail System einsetzen können, sollten Sie sich etwasit den (lesbaren) RFC's befassen.

Als Vorübung senden und empfangen wir emails direkt mit Telnet. Da wir aber Java einsetzen, ersetzen wir Telnet durch eine Socket-Verbindung. Hier eine typische Telnet Session, um emails in einer Mailbox aufzulisten (das Beispiel stammt mehr oder weniger direkt aus dem RFC) :

1.4.1. Abfragen

```
telnet pop3.host.com 110
+OK Hello there.
USER mail_user (Ihr Benutzername)
+OK Password required.
PASS mail_user_pwd (Ihr Passwort für den Mail Server)
+OK logged in.
LIST
…
.
```

1.4.2. Senden

```
telnet smtp.host.com
220 smtp.hispeed.ch ESMTP Sendmail 8.12.6/8.12.6/tornad
HELO Klaus
250 smtp.hispeed.ch Hello <Ihr Rechnername>
mail from: user@host.com
250 2.1.0 user@host.com... Sender ok
rcpt to:joller@joller-voss.ch
250 2.1.5 joller@joller-voss.ch... Recipient ok
data
354 Enter mail, end with "." on a line by itself
subject: Telnet Mail
Hallo jo
Wie geht's
.
250 2.0.0 i26F9KSk015323 Message accepted for delivery
```

1.4.3. emails über Sockets empfangen

Nun machen wir dasselbe wie in Telnet mithilfe einer Socketverbindung:

```
/*
 * Created on 29.02.2004
 * TelnetMail
 *
*/
package joller;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.Socket;
import java.net.UnknownHostException;

/**
 * @author jjoller
 *
 */
public class TelnetMailReceive {
	private final String POP3_HOST = "pop3.host";
	//    Beispiel    "mail.joller-voss.ch"; // eingehend
	private final int POP3_PORT = 110;
	private final boolean DEBUG = true;
	public static void main(String[] args) {
		TelnetMailReceive tel = new TelnetMailReceive();
		String line = "";
		String match = "";
		try {

			// Senden
			Socket sok = new Socket(tel.POP3_HOST, tel.POP3_PORT);

			// einloggen in die Mailbox
			BufferedWriter bw =
				new BufferedWriter(
				new OutputStreamWriter(sok.getOutputStream()));

			// Antwort
			BufferedReader br =
				new BufferedReader(
				new InputStreamReader(sok.getInputStream()));
			line="\r\n";
			if (tel.DEBUG) System.out.print(line);
			bw.write(line);

			line="";
			line=br.readLine();
			System.out.println(line);

			// UID
			line = "USER benutzername";
			// Beispiel "USER ich@joller-voss.ch\r\n";
			if (tel.DEBUG) System.out.print(line);
			bw.write(line);
			bw.flush();
```

```
                    // Antwort vom Mail Server
                    line="";
                    match = "password please";
                    while ((line=br.readLine()).indexOf(match)<0) {
                          System.out.println(line);
                    }

                    // PWD
                    line = "PASS mein_passwort";
                    // Beispiel "PASS xyz\r\n";
                    if (tel.DEBUG) System.out.print(line);
                    bw.write(line);
                    bw.flush();

                    // Antwort vom Mail Server
                    line="";
                    match = "+OK";
                    while((line=br.readLine()).indexOf(match)<0) {
                          System.out.println(line);
                    }

                    // Mailbox lesen
                    line = "LIST\r\n";
                    if (tel.DEBUG) System.out.print(line);
                    bw.write(line);
                    bw.flush();

                    line="";
                    match =".";
                    do {
                          line=br.readLine();
                          System.out.println(line);
                    } while(line.indexOf(match)<0);

                    System.out.println("[ENDE]");

              } catch (UnknownHostException e) {
                    System.err.println(tel.POP3_HOST + " nicht gefunden");
                    e.printStackTrace();
              } catch (IOException e) {
                    System.err.println(
                          "Fehler bei der Verbindung mit " + tel.POP3_HOST);
                    e.printStackTrace();
              }
        }
}
```

Beachten Sie, dass beim Schreiben unbedingt jeweils der Buffer geleert werden muss. Sonst wartet der Server auf weitere Eingaben und Sie staunen, dass nichts passiert.

Der Rest des Programms ist mehr oder weniger selbsterklärend. Die Abfragen der Antworten des Servers könnten sich in Ihrem Beispiel unterscheiden. Ich hielt mich an den Standard.

Bei Problemen können Sie immer noch auf die Telnet Session ausweichen und sich die Antworten merken und ins obige Programm einbauen.

Beachten Sie auch, dass der Absender nicht validiert wird: es wird alles akzeptiert.

1.4.4. emails über Sockets senden

Nun senden wir eine Meldung wie in Telnet mithilfe einer Socketverbindung:

```
/*
 * Created on 29.02.2004
 * TelnetMail
 *
*/
package joller;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.Socket;
import java.net.UnknownHostException;

/**
 * @author jjoller
 *
 */
public class TelnetMailSend {
	private final String HOST = "mail.host";
	// Beispiel "smtp.swissonline.ch";
	private int PORT = 25;
	private final String TO = "mail_receiver@host";
	//Beispiel "ich@joller-voss.ch";
	private final String FROM = "mail_sender@host";
	//Beispiel "josef.m.joller@swissonline.ch";
	private final boolean DEBUG = true;
	public static void main(String[] args) {
		TelnetMailSend tel = new TelnetMailSend();
		String line = "";
		String match = "";
		try {
			Socket sok = new Socket(tel.HOST, tel.PORT);

			BufferedWriter bw =
				new BufferedWriter(
				new OutputStreamWriter(sok.getOutputStream()));
			BufferedReader br =
				new BufferedReader(
				new InputStreamReader(sok.getInputStream()));
			line = "Helo " + tel.HOST + "\r\n";
			if (tel.DEBUG)
				System.out.println("[HELO]" + line);
			bw.write(line);
			bw.flush();
			match = "pleased to meet you";
			while ((line = br.readLine()).indexOf(match) <0) {
				if (tel.DEBUG)
					System.out.println("[HELO]" + line);
			}

			line = "MAIL FROM:<" + tel.FROM + ">\r\n";
			if (tel.DEBUG)
				System.out.println("[MAIL FROM]" + line);
			bw.write(line);
			bw.flush();
			line = "";
			match = "ok";
```

```
            while ((line = br.readLine().toLowerCase()).indexOf(match)<0) {
                  if (tel.DEBUG)
                        System.out.println("[MAIL FROM]" + line);
            }

            line = "RCPT TO:" + tel.TO + "\r\n";
            if (tel.DEBUG)
                  System.out.println("[RCPT]" + line);
            bw.write(line);
            bw.flush();
            line = "";
            match = "ok";
            while ((line = br.readLine().toLowerCase()).indexOf(match)<0) {
                  System.out.println("[RCPT]" + line);
            }

            line = "DATA\r\n";
            if (tel.DEBUG) System.out.println("[DATA]" + line);
            bw.write(line);
            bw.flush();
            line = "";
            match = "itself";
            while ((line = br.readLine().toLowerCase()).indexOf(match)<0) {
                  if (tel.DEBUG)
                        System.out.println("[DATA]" + line);
            }
            line = "Subject:Test\r\n"+
                        "Test!\r\n"+"\r\n"+
                        "."+"\r\n"+"QUIT\r\n";
            if (tel.DEBUG) System.out.println("[DATA]" + line);
            bw.write(line);
            bw.flush();
            bw.close();
            System.out.println("[ENDE_MAIL]");

            } catch (UnknownHostException e) {
                  System.err.println(tel.HOST + " nicht gefunden");
                  e.printStackTrace();
            } catch (IOException e) {
                  System.err.println("Fehler bei der Verbindung mit " +
                                    tel.HOST);
                  e.printStackTrace();
            }
      }
}
```

Beachten Sie, dass das Subject in der Data Sektion angegeben wird. In der Regel ist Gross- und Kleinschreibung unwichtig.

Auch hier können Sie, falls etwas nicht klappt, einfach mit Telnet nachschauen, was nicht stimmt (in Ihrem Fall, auf Ihrem Mail Server) und dann das obige Programm anpassen.

1.5. Übung

Setzen Sie Ihre JavaMail Umgebung auf.

1.5.1. Installation der JavaMail Referenz Implementation

Lernziele:

- Aufsetzen der Umgebung, um JavaMail Applikationen zu übersetzen und zu starten.
- Testen eines Demo Programms, welche mit der Referenz Implementation geliefert wird.

In dieser einführenden Übung installieren Sie die Sun JavaMail Referenz Implementation und starten eines der Beispielprogramme, welche mitgeliefert werden.

1.5.2. Aufgabe - Herunterladen von JavaMail

http://java.sun.com/products/javamail/ ist die Quelle.
Sie können aber auch beispielsweise ab der Java JumpStart CD (CD2) installieren

1.5.3. Aufgabe - Herunterladen des JavaBeans Activation Frameworks

http://java.sun.com/products/javabeans/glasgow/jaf.html ist die Quelle.
Sie können aber auch beispielsweise ab der Java JumpStart CD (CD2) installieren

1.5.4. Aufgabe - Unzip

Speichern Sie die Dateien in einem passenden Verzeichnis, beispielsweise ab Root C:. Sie können dazu WinZIP oder das Java Jar Programm verwenden.

1.5.5. Aufgabe - Archive in den CLASSPATH eintragen

Die Archive `mail.jar` und `activation.jar` (JavaBeans Activation Framework) und eventuell `pop3.jar` müssen im CLASSPATH stehen oder sonstwie erreichbar sein.
Alternative:

```
cd \javamail-1.3.1
copy mail.jar %JAVA_HOME%\jre\lib\ext
cd \jaf-1.0.1
copy activation.jar %JAVA_HOME%\jre\lib\ext
Rem eventuell pop3.jar
```

1.5.6. Aufgabe - Installationstest

Wechseln Sie ins Verzeichnis `demo` Ihre JavaMail API Installation. Übersetzen Sie `msgsend` und senden Sie eine Testmeldung.

```
javac msgsend.java <Optionen>
```

Beachten Sie die Optionen des Programms:
-o : from@address (Originator)
-M SMTP-Server (Mailserver)
to@address (Empfänger)

```
java msgsend –o from@address –M SMTP.Server to@address
```

also

```
java msgsend –o josef.m.joller@swissonline.ch –M smtp.swissonline.ch
alma.matter@uni.edu
```

Prüfen Sie mit Ihrem Standard Mail Programm, ob die Nachricht ankommt.

1.5.7. Lösung

Falls Sie Ihr Mail empfangen haben, ist alles korrekt aufgesetzt. Falls Sie eine `Class not found` Meldung erhalten, stimmt Ihr CLASSPATH nicht!

1.5.8. Demonstration

Hier eine Beispiel Session:

```
------------------------------------------------------------------
Message Send : Demo Applikation von JavaMail (C) Sun Microsystems
------------------------------------------------------------------
Schritt 1: uebesetzen
Press any key to continue . . .
To: joller@joller-voss.ch
Subject: Test JavaMail 2004
Activation Framework muss im CLASSPATH sein

.
^Z

Mail was sent successfully.
```

Beachten Sie, dass Sie unbedingt die Libraries im Classpath haben müssen (`mail.jar, pop3.jar,activation.jar`):

```
@echo off
@echo ------------------------------------------------------------------
@echo Message Send : Demo Applikation von JavaMail (C) Sun Microsystems
@echo ------------------------------------------------------------------
@echo Schritt 1: uebesetzen
Rem ------------------------------------------------------------------
Rem ich habe JAVA_MAIL als Umgebungsvariable auf C:\javamail-1.3.1
Rem und JAVA_ACTIVATION auf c:\jaf-1.0.2 gesetzt
Rem ------------------------------------------------------------------

set JAVA_MAIL=c:\javamail-1.3.1
set JAVA_ACTIVATION=C:\jaf-1.0.2
set
CLASSPATH=..\mail.jar;..\lib\mailapi.jar;..\lib\imap.jar;..\lib\pop3.jar;..
\smtp.jar;.;..;..\..\jaf-1.0.2\activation.jar
@echo ------------------------------------------------------------------
@echo Schritt 2: Mail versenden
%JAVA_HOME%\bin\javac -classpath %CLASSPATH% msgsend.java
pause
Rem ------------------------------------------------------------------
Rem
Rem o = originator
Rem m = mailhost
Rem
java msgsend -o joller@joller-voss.ch -M smtp.swissonline.ch joller@joller-
voss.ch
```

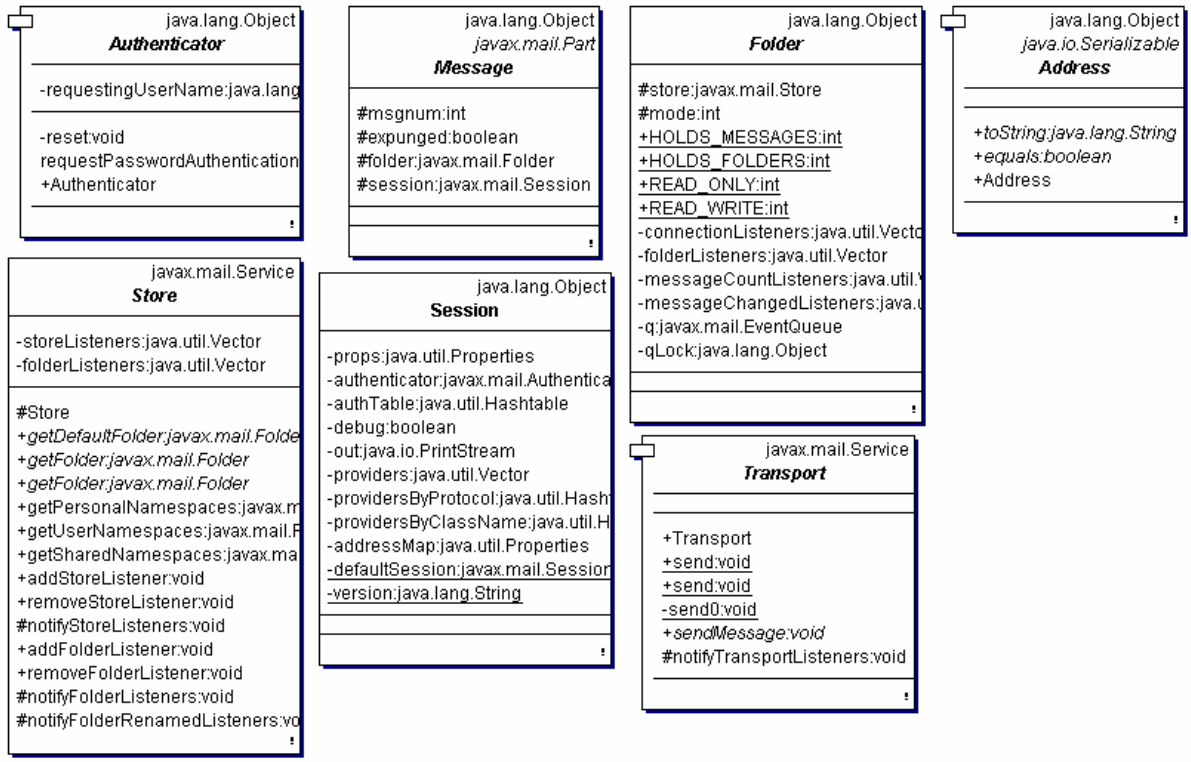
Falls Sie die Programme im JBuilder zum Laufen kriegen, aber lieber eine Batch Datei hätten, können Sie die erste Ausgabezeile des laufenden Programms im JBuilder in eine Batch Datei kopieren. Diese beschreibt das Starten der JVM mit allen korrekten Parametern.

Auf dem Server finden Sie Batch Dateien und alle Beispiele mit Lösungen als Eclipse Workspace (als Tomcat Projekt: wir wollen auch Servlets anbinden).

1.6. Review der Core Klassen

Bevor wir uns mit dem "How to ... " befassen, wollen wir zuerst einen Blich in die Klassen werfen, die das JavaMail API ausmachen:

- `Session`,
- `Message`,
- `Address`,
- `Authenticator`,
- `Transport`,
- `Store` und
- `Folder`.



Diese Klassen sind Teil des Top Level Packages vom JavaMail API: `javax.mail`.
Oft werden Sie allerdings Unterklassen einsetzen, beispielsweise aus dem `javax.mail.internet` Package (siehe Demo Programm).

Das gesamte Klassendiagramm ist zu komplex, um hier angezeigt zu werden.

1.6.1. Session

Die Session Klasse definiert eine Mail Session. Dies ist der Grundbaustein auf dem alles aufsetzt. Das `Session` Objekt verwendet `java.util.Properties` Objekte, um Informationen wie zum Beispiel den Mail Server, den Benutzer und dessen Passwort und andere Informationen, die von verschiedenen Applikationen benutzt werden, zu erhalten.

Die Konstruktoren dieser Klasse sind `private`. Eine Standard Session erhält man mittels der `getDefaultInstance()` Methode:

```
Properties props = new Properties();
// props enthält alle relevanten Informationen
// ...
Session session =
Session.getDefaultInstance(props, null);
```

Alternativ dazu können Sie auch `getInstance()` einsetzen:

```
Properties props = new Properties();
// props enthält alle relevanten Informationen
Session session = Session.getInstance(props,
null);
```

In beiden Fällen wird das `null` Objekt als `Authenticator` Objekt eingesetzt, da wir es zur Zeit nicht benötigen, mehr darüber erfahren Sie weiter unten.

In den meisten Fällen genügt es die Shared Session einzusetzen, selbst wenn man mit Mail Sessions für mehrere User Mailboxen arbeitet. Benutzername und Passwort kann man später immer noch ergänzen.

1.6.2. Message

Nachdem wir ein `Session` Objekt haben, müssen wir Nachrichten kreieren und verschieben (lesen und senden). Dies geschieht mit `Message` Objekten. Diese Klasse ist abstrakt, Sie müssen also mit den Unterklassen dieser Klasse arbeiten. In den meisten Fällen wird dies `javax.mail.internet.MimeMessage` sein. Eine `MimeMessage` ist eine email Message, welche einen bestimmten MIME Type erkennt, gemäss RFC. Message Headers bestehen aus US-ASCII Zeichen, nicht-ASCII Zeichen können lediglich in bestimmten Headerfeldern verwendet werden.

Um eine `Message` zu kreieren wird das `Session` Objekt an den `MimeMessage` Konstruktor übergeben:

```
MimeMessage message = new MimeMessage(session);
```

Bemerkung: Es gibt auch noch andere Konstruktoren, beispielsweise für das Kreieren von gemäss RFC822-formattierten Input Streams.

Nachdem Sie ein `Message` Objekt haben, können Sie dessen Teile, Parts spezifizieren. Die `Message` Klasse implementiert das `Part` Interface, `MimeMessage` implementiert das `MimePart` Interface. Der grundlegende Mechanismus den Inhalt einer Message zu setzen besteht aus einem Methodenaufruf von `setContent()` mit dem Message Inhalt als Argument: plus MIME Type:

```
message.setContent("Meine erste Nachricht an die
Welt: bäh", "text/plain");
```

Falls bekannt ist, dass man es mit einer `MimeMessage` zu tun hat, dann besteht die Meldung lediglich aus Text und Sie können genauso die Methode `setText()` einsetzen. Diese berwendet einfach als Standardwert den MIME Type text/plain:

```
message.setText("Hello");
```

Falls Sie lediglich einfache Textmeldungen verschicken wollen, verwenden Sie sinnvollerweise die letztere Form.

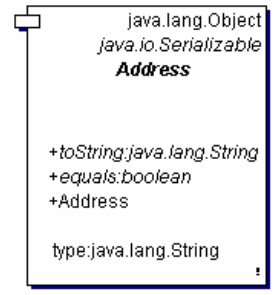
Falls Sie eine HTML Meldung versenden wollen, verwenden Sie besser die erste Methode! Warum dies so ist, werden wir weiter unten sehen.

Das 'Betreff' oder 'Subject' Feld setzen Sie mit der `setSubject()` Methode:

```
message.setSubject("wie versprochen...");
```

1.6.3. Address

Nachdem Sie nun das `Session` und das `Message` Objekt konstruiert haben und die Meldung, den Inhalt der Nachricht definiert haben, müssen Sie diese an jemanden adressieren. Dies geschieht mittels eines `Address` Objekts. Auch diese Klasse ist abstrakt! Sie können beispielsweise die Unterklasse `javax.mail.internet.InternetAddress` Klasse verwenden.



Um eine Adresse zu konstruieren übergeben Sie einfach die email Adresse an den Konstruktor:

```
Address address = new
InternetAddress("president@whitehouse.gov");
// wirft keine Exception
```

Falls Sie möchten, dass neben der email Adresse auch noch ein Namen erscheint, können Sie diesen gleich dem Konstruktor mitgeben:

```
Address address = new InternetAddress("ceo@firma.com", "Der Boss");
```

Adressen müssen Sie sowohl für das *from / von* als auch für das *to / an* Feld konstruieren. Der Mailserver hindert Sie aber nicht daran einen falschen Sender anzugeben.

Nachdem Sie die Adressen konstruiert haben, müssen Sie diese mit der Nachricht verknüpfen. Dies kann auf eine von zwei Arten geschehen:

`setFrom()` bzw.
`setReplyTo()` und
`message.setFrom(address)`

definieren den Absender.
Falls Sie mehrere Absender anbegen möchten, können Sie weitere mit der `addFrom()` Methode hinzufügen:

```
Address address[] = ...;
message.addFrom(address);
```

Den Empfänger geben Sie mit der Methode `addRecipient()` an. Diese Methode verlangt einen `Message.RecipientType` neben der Adresse:

```
message.addRecipient(type, address)
```

Drei mögliche Typen sind möglich:

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

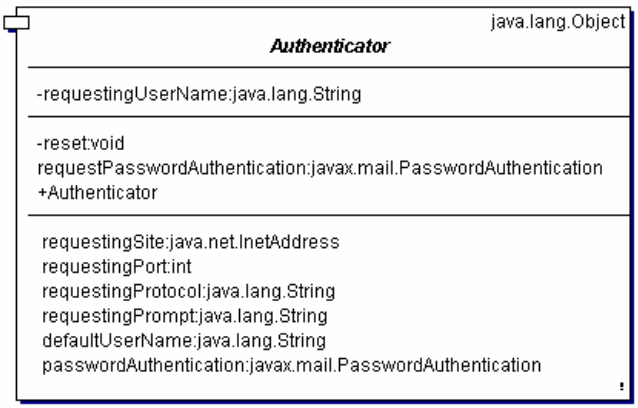
Hier ein vollständiges Beispiel:

```
Address toAddress = new InternetAddress("besitzer@firma.com");
Address ccAddress = new InternetAddress("shareholder@firma.com");
message.addRecipient(Message.RecipientType.TO, toAddress);
message.addRecipient(Message.RecipientType.CC, ccAddress);
```

JavaMail stellt keinerlei Mechanismen zur Überprüfung der Mail Adressen zur Verfügung. Sie können aber im Programm überprüfen, ob irgendwelche Sonderzeichen gemäss RFC 822 vorkommen oder auch den MX Record (Mail Exchange Record im DNS) abfragen. Aber JavaMail erledigt nichts für Sie.

1.6.4. Authenticator

Wie in den `java.net` Klassen hat das JavaMail API den Vorteil, dass ein `Authenticator` Objekt den Zugriff auf geschützte Ressourcen überwachen kann, mittels Benutzernamen und Passwort. Im Falle von JavaMail handelt es sich dabei um den Mail Server. Die `Authenticator` Klasse befindet sich im JavaMail API im `javax.mail` Package und unterscheidet sich von der entsprechenden `java.net` Klasse in einigen Details. Beispielsweise funktioniert die JavaMail Version auch im JDK 1.1 Umfeld, die Netzwerkversion aber noch nicht!



Konkret einsetzen können Sie die Klasse, indem Sie eine Unterklasse bilden oder verwenden, beispielsweise `PasswordAuthentication` Die Methode `getPasswordAuthentication()` überprüft die Authentifizierung. Sie müssen das `Authenticator` Objekt beim Kreieren der Session angeben. Immer wenn eine Authentifizierung nötig wird, wird in diesem Fall dieses Objekt eingesetzt. Sie könnten genauso ein PopUp Fenster definieren und den Namen und das Passwort jeweils abfragen. Alternativ dazu können Sie die Berechtigungen aus einer Property Datei lesen, wobei die Verschlüsselung allerdings problematisch wird. Damit könnten Sie eine Passwort Authentifizierung einbauen:

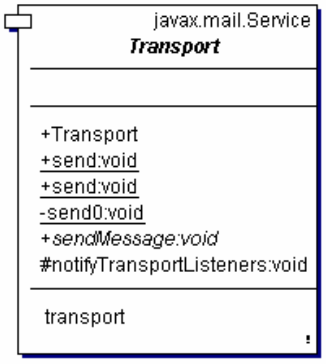
```
Properties props = new Properties();
// props Infos bestimmen ...
Authenticator auth = new PopupAuthenticator ();
Session session = Session.getDefaultInstance(props, auth);
```

Die Authenticator Klasse ist dabei folgendermassen definiert:

```
public class PopupAuthenticator extends Authenticator {

      public PasswordAuthentication getPasswordAuthentication() {
            String username, password;

            String result =
                  JOptionPane.showInputDialog("'username,password'");

            StringTokenizer st = new StringTokenizer(result, ",");
            username = st.nextToken();
            password = st.nextToken();

            return new PasswordAuthentication(username, password);
      }
}
```

1.6.5. Transport

Am Schluss bleibt uns nur noch das Senden der Nachricht mittels der `Transport` Klasse. Diese Klasse verwendet bestimmte Protokolle, beispielsweise SMTP, um Nachrichten zu versenden. Die Klasse ist wie üblich abstrakt und funktioniert ähnlich wie die `Session` Klasse.



Falls Sie keine speziellen Anforderungen haben, können Sie das Standardprotokoll verwenden und einfach die static `send()` Methode einsetzen:

```
Transport.send(message);
```

In allen anderen Fällen müssen Sie beispielsweise den Transport bestimmen, der einer Session zugeordnet ist:

```
message.saveChanges(); // implicit mit send()
Transport transport = session.getTransport("smtp");
// Standard
transport.connect(host, username, password);
transport.sendMessage(message, message.getAllRecipients());
transport.close();
```

Falls Sie mehrere Meldungen senden müssen, können Sie mit der letzteren Formulierung die Anzahl Sessions optimieren, Sie müssen also nicht dauernd neue Verbindungen zum Mail Server aufbauen. Beim Einsatz den `send()` geschieht bei jedem Aufruf ein neuer Verbindungsaufbau. Mit `session.setDebug(true)` können Sie die Befehle, die an den Mail Server gesendet werden verfolgen:

```
[Senden]main()
DEBUG: setDebug: JavaMail version 1.3.1
DEBUG: getProvider() returning
javax.mail.Provider[TRANSPORT,smtp,com.sun.mail.smtp.SMTPTransport,Sun
Microsystems, Inc]
DEBUG SMTP: useEhlo true, useAuth false
DEBUG SMTP: trying to connect to host "smtp.swissonline.ch", port 25

220 smtp.hispeed.ch ESMTP Sendmail 8.12.6/8.12.6/tornado-1.0; Sat, 6 Mar
2004 21:40:56 +0100
DEBUG SMTP: connected to host "smtp.swissonline.ch", port: 25

EHLO NINFPDW11
250-smtp.hispeed.ch Hello 217-162-161-242.dclient.hispeed.ch
[217.162.161.242], pleased to meet you
250-ENHANCEDSTATUSCODES
250-PIPELINING
250-8BITMIME
250-SIZE 10485760
250-DSN
250-STARTTLS
250-DELIVERBY
250 HELP
DEBUG SMTP: Found extension "ENHANCEDSTATUSCODES", arg ""
DEBUG SMTP: Found extension "PIPELINING", arg ""
DEBUG SMTP: Found extension "8BITMIME", arg ""
DEBUG SMTP: Found extension "SIZE", arg "10485760"
DEBUG SMTP: Found extension "DSN", arg ""
DEBUG SMTP: Found extension "STARTTLS", arg ""
```

```
DEBUG SMTP: Found extension "DELIVERBY", arg ""
DEBUG SMTP: Found extension "HELP", arg ""
DEBUG SMTP: use8bit false
MAIL FROM:<absender@joller-voss.ch>
250 2.1.0 <absender@joller-voss.ch>... Sender ok
RCPT TO:<joller@joller-voss.ch>
250 2.1.5 <joller@joller-voss.ch>... Recipient ok
DEBUG SMTP: Verified Addresses
DEBUG SMTP:   joller@joller-voss.ch
DATA
354 Enter mail, end with "." on a line by itself
Message-ID: <27994366.1078605654591.JavaMail.jjoller@NINFPDW11>
From: absender@joller-voss.ch
To: joller@joller-voss.ch
Subject: Hallo aus JavaMail
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit

Willkommen bei JavaMail
.
250 2.0.0 i26KeufM028171 Message accepted for delivery
QUIT
[Senden.main()]Ende
```

1.6.6. Store und Folder

Das Lesen einer Meldung geschieht analog zum Senden, mit Hilfe des `Session` Objekts. In diesem Fall muss man aber in der Regel nach dem man eine Session eröffnet hat, sich authentifizieren, da man nicht ohne Prüfung auf einen Mail Server lesend zugreifen kann, in JavaMail Terminologie: einen Speicher, oder ein `Store` Objekt. In Java können Sie dies mittels eines `Authenticator` Objekts realisieren (Alternativen finden Sie in den Beispielen).

Wie beim `Transport` teilen Sie dem `Store` mit, welches Protokoll Sie dazu verwenden möchten:

```
// Store store = session.getStore("imap"); //
falls Sie IMAP verwenden
Store store = session.getStore("pop3");
// falls Sie POP3 verwenden
store.connect(host, username, password);
```

Nach dem Verbindungsaufbau zum `Store` erhalten Sie ein `Folder` Objekt, welches geöffnet werden muss, bevor Sie Meldungen daraus lesen können:


```
Folder folder = store.getFolder("INBOX");
folder.open(Folder.READ_ONLY);
Message message[] = folder.getMessages();
```

Im Falle von POP3 existiert lediglich ein Folder, die `INBOX`.
Falls Sie IMAP einsetzen können viele weitere, von Ihnen definierte Folder existieren. Sie können auch `POP3Folder` kreieren.

Nachdem Sie eine Meldung, ein `Message` Objekt gelesen haben, können Sie seinen Inhalt mit `getContent()` bestimmen oder seinen Inhalt in einen Stream stellen.: `writeTo()`. Die `getContent()` Methode liest lediglich den Inhalt der Message. Die Methode `writeTo()` schreibt auch noch die Headerinformation raus.

```
System.out.println(((MimeMessage)message).getContent()
);
```

Wie bei fast allen Programmen, die auf Ressourcen zugreifen, die geshared werden, sollten Sie möglichst nach dem Lesen die Folders wieder schliessen:

```
folder.close(aBoolean);
store.close();
```

Mit der Boole'schen Variable bestimmen Sie, ob bei der `close()` Methode der Folder modifiziert werden soll, also gelöschte Einträge entfernt werden sollen.

1.6.7. Wie geht's weiter?

Nun haben wir die Übersichtstour abgeschlossen und wollen uns mit konkreten Beispielen von JavaMail beschäftigen. Einzelne Aufgaben, wie etwa das Durchsuchen mittels Schlüsselwörtern, werden wir später noch anschauen.

1.7. JavaMail API Praxis

Bisher haben wir die Kernklassen das JavaMail API kennen gelernt. In den folgenden Abschnitten wollen wir nun mit diesen Klassen Applikationen für bestimmte Aufgaben zusammenbauen.

1.7.1. Senden von Messages

Das Versenden von emails besteht aus mehreren Schritten:

1. kreieren einer Session,
2. kreieren einer Nachricht
3. versenden einer Nachricht

Falls Sie Ihren SMTP Server als Property angeben, mittels der `mail.smtp.host` Property, können Sie mit einem `Properties` Objekt diese Information ins Programm übernehmen und an das `Session` Objekt übergeben:

```
public class Senden {
  public static void main (String args[]) throws Exception {
    System.out.println("[Senden]main()");
    String setup[]= {"smtp.swissonline.ch", "absender@joller-voss.ch",
"joller@joller-voss.ch"};
    /* String host = args[0];String from = args[1]; String to = args[2]; */
    String host = setup[0];
    String from = setup[1];
    String to = setup[2];

    // 1) System Properties lesen
    Properties props = System.getProperties();

    // 2) Definition des Mail Server
    props.put("mail.smtp.host", host);

    // 3) Kreieren einer Session
    Session session = Session.getDefaultInstance(props, null);
    session.setDebug(true);

    // 4) festlegen der Message
    MimeMessage message = new MimeMessage(session);

    // 4.a) from Adresse
    message.setFrom(new InternetAddress(from));

    // 4.b) to Adresse
    message.addRecipient(Message.RecipientType.TO,
      new InternetAddress(to));

    // 4.c) Subject
    message.setSubject("Hallo aus JavaMail");

    // 4.d) Inhalt / Content
    message.setText("Willkommen bei JavaMail");

    // 5) Senden der Message
    Transport.send(message);
      System.out.println("[Senden.main()]Ende");
  }
}
```

1.7.2. Übung - Senden einer Meldung

In der letzten Übung haben Sie eine Mail Message mehr oder weniger von Hand versendet. In dieser Übung übernimmt das Java Programm die Arbeit für Sie. Ziel der Übung ist es, dass Sie den Programmcode, basierend auf einem Gerüst, selber schreiben.

1.7.2.1. Voraussetzungen

Sie müssen Ihr JavaMail bereits installiert und getestet haben, wie in der ersten Übung beschrieben.

1.7.2.2. Lernziele

In dieser Übung lernen Sie ein einfaches JavaMail Programm zu schreiben, welches eine einfache Text- Mitteilung an einen Empfänger senden.

Lernziele:

- Sie lernen die `Session` Klasse kennen
- Sie können eine `MimeMessage` Message kreieren
- Sie instanzieren die Klasse `InternetAddress`, umIhre Meldung zu adressieren (von / an)
- Sie versenden Messages unter zu Hilfenahme der `Transport` Klasse

1.7.2.3. Programmskizze

```
public class Senden {
      public static void main(String args[]) throws Exception {
            String host = args[0];
            String from = args[1];
            String to = args[2];

            // bestimme System Properties

            // Setup Mail Server

            // Session festlegen

            // Definiere Message

            // 'from' Adresse festlegen

            // 'to' Adresse festlegen

            // Subject definieren

            // Content eingeben

            // Senden der Message
      }
}
```

Diese erste Übung sollten Sie schaffen, falls Sie das obige Beispiel angeschaut haben!

1.7.2.4. Aufgabe - Bestimmen Sie die System Properties

Lösung:

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

1.7.2.5. Aufgabe - Fügen Sie mail.smtp.host hinzu

Lösung:

```
props.put("mail.smtp.host", host);
```

1.7.2.6. Kreieren eines Session Objekts (Eigenschaften in props)

Lösung:

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

1.7.2.7. Kreieren Sie eine MimeMessage für dieses Session Objekt

Lösung:

```
MimeMessage message = new MimeMessage(session);
```

1.7.2.8. Definieren Sie das 'from' Feld

Lösung:

```
message.setFrom(new InternetAddress(from));
```

1.7.2.9. Definieren Sie das 'to' Feld

Lösung:

```
message.addRecipient(Message.RecipientType.TO, new InternetAddress(to));
```

1.7.2.10. Definieren Sie das 'subject' Feld

Lösung:

```
message.setSubject("Hallo JavaMail");
```

1.7.2.11. Definieren des Inhalts der Meldung

Lösung:

```
message.setText("Ein Gruss aus JavaMail");
```

1.7.2.12. Versenden Sie die Nachricht mit Transport

Lösung:

```
Transport.send(message);
```

1.7.2.13. Übersetzen und Starten Sie das Programm

Dazu müssen Sie noch den SMTP Server festlegen, die 'to', 'from' Felder / Adressen initialisieren und los geht's!

Lösung:

Sie finden die Musterlösung auf dem Server / der CD

1.7.2.14. Musterlösung

```
package praxis;

import java.util.Properties;
import javax.mail.Message;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;

/**
 * Versenden einer einfachen Nachricht mit JavaMail
*/
public class Senden {
      public static void main(String args[]) throws Exception {
            System.out.println("[Senden]main()");
            String setup[] =
                  {
                        "smtp.swissonline.ch",
                        "absender@joller-voss.ch",
                        "joller@joller-voss.ch" };
            /* String host = args[0];String from = args[1];
            String to = args[2]; */
            String host = setup[0];
            String from = setup[1];
            String to = setup[2];

            // 1) System Properties lesen
            Properties props = System.getProperties();

            // 2) Definition des Mail Server
            props.put("mail.smtp.host", host);

            // 3) Kreieren einer Session
            Session session = Session.getDefaultInstance(props, null);
            session.setDebug(true);

            // 4) festlegen der Message
            MimeMessage message = new MimeMessage(session);

            // 4.a) from Adresse
            message.setFrom(new InternetAddress(from));

            // 4.b) to Adresse
            message.addRecipient(Message.RecipientType.TO,
                                       new InternetAddress(to));

            // 4.c) Subject
            message.setSubject("Hallo aus JavaMail");

            // 4.d) Inhalt / Content
            message.setText("Willkommen bei JavaMail");

            // 5) Senden der Message
            Transport.send(message);
            System.out.println("[Senden.main()]Ende");
      }
}
```

1.7.2.15. Demonstration

Falls Sie die args[...] Version benutzen, können Sie durch folgenden Aufruf eine Kurznachricht an Ihren Mail Account schicken:

```
java Senden smtp.swissonline.ch sender@host.com joller@joller-voss.ch
```

und nach kurzer Zeit sehe ich die Meldung in meiner Inbox.

1.7.3. Lesen von Nachrichten

Damit Sie Meldungen lesen können, benötigen Sie eine Session und müssen eine Verbindung zu einem Store, Ihrer Mailbox aufbauen, den passenden Folder öffnen und Ihre Nachricht(en) lesen.

Nach dem Lesen sollten Sie das Schliessen nicht vergessen!

```
/*
  String host = args[0]; String username = args[1];
  String password = args[2];
*/
String setup[] =
{"mail.joller-voss.ch", "UID", "PWD"};
// 1) Properties bestimmen oder setzen
Properties props = new Properties();
String host = setup[0];
String username = setup[1];
String password = setup[2];

// 1.a) System Properties bestimmen (Argumente)
// Properties props = System.getProperties();

// 2) Mailbox Server festlegen
props.put("mail.pop3.host", host);

// 3) Session kreieren
Session session = Session.getDefaultInstance(props, null);
session.setDebug(true);

// 4) Mailbox festlegen (store)
Store store = session.getStore("pop3");

// 4.1) Verbindung zur Mailbox aufbauen
store.connect(host, username, password);

// 4.2) Folder / Mailbox bestimmen
Folder folder = store.getFolder("INBOX");

// 4.3) Mailbox read-only (lesend) öffnen
folder.open(Folder.READ_ONLY);

BufferedReader reader =
      new BufferedReader(new InputStreamReader(System.in));

// 5) Directory bestimmen
javax.mail.Message[] message = folder.getMessages();
for (int i = 0, n = message.length; i < n; i++) {
      // 6) Inhalt
      // 6.1) von Subject und From Feld anzeigen
      System.out.println(
                  + message[i].getFrom()[0]
```

```
                        + "\t"
                        + message[i].getSubject());
                  // 6.2) Message Inhalt anzeigen
                  System.out.println(message[i].getContent());
            }
            folder.close(false);
            store.close();
```

Das sieht sehr einfach aus und ist es auch.Was Sie mit den Meldungen machen ist Ihnen überlassen! Das obige Rahmenprogramm zeigt einfach die Felder 'from' und 'subject' der Meldungen an. Im schlimmsten Fall wurde keine 'from' Adresse angegeben. *In diesem Fall würde eine Exception geworfen.*

Falls Sie die ganze Nachricht ansehen wollen, müssen Sie diese auch mit der `writeTo()` Methode ausgeben:

```
            // 6) Inhalt
            // 6.1) von Subject und From Feld anzeigen
            System.out.println(
                  i
                        + ": "
                        + message[i].getFrom()[0]
                        + "\t"
                        + message[i].getSubject());

            System.out.println(
                  "Moechten Sie die Nachricht lesen?
                  [JA zum Lesen/QUIT falls nicht]");
            String line = reader.readLine();
            if ("JA".equals(line)) {
                  // 6.2) Message Inhalt anzeigen
                  System.out.println(message[i].getContent());
            } else if ("QUIT".equals(line)) {
                  break;
            }
```

1.7.4. Übung - Prüfen ob neue Nachrichten eingetroffen sind

In dieser Übung kreieren Sie ein Programm, mit dem Sie das 'from' und 'subject' Feld der Meldungen in der Mailbox ausgeben und fragen, ob die Meldung als Ganzes ausgegeben werden soll (siehe oben).

1.7.4.1. Voraussetzungen

Ihr JavaMail muss korrekt aufgesetzt sein!

1.7.4.2. Lernziele

In dieser Übung benutzen Sie das JavaMail API, um Nachrichten zu lesen.
Die Lernziele im Einzelnen:

- Sie lernen die `Store` Klasse kennen
- Sie können einen `Folder` eines Stores bestimmen
- Sie lesen ein `Message` Objekt aus einem Folder

1.7.4.3. Programmskizze

```
package praxisvorgabe;

import java.io.BufferedReader;
import java.io.InputStreamReader;

public class Lesen {
      public static void main(String args[]) throws Exception {
            String host = args[0];
            String username = args[1];
            String password = args[2];

            // 1) Kreie leere Properties

            // 2) Instanziere eine Session

            // 3) Store / Mailbox Handlimg

            // 4) Connect zum Store

            // 5) Folder

            // 6) Öffnen (read-only) : lesend

            BufferedReader reader =
                  new BufferedReader(new InputStreamReader(System.in));

            // 7) Ausgabe der Mail-Liste

            /** die folgenden Zeilen stellen eine mögliche Lösung dar
             * für die Ausgabe des Message Inhalts
             */
//          for (int i = 0, n = message.length; i < n; i++) {
//
//                // 8) Anzeige von 'from' und 'subject'
//
//                System.out.println(
//                      "Moechten Sie die Nachricht lesen? [JA falls Sie
lesen wollen/QUIT sonst]");
```

```
//                String line = reader.readLine();
//                if ("JA".equals(line)) {
//
//                      // 9) Message Content anzeigen
//
//                } else if ("QUIT".equals(line)) {
//                      break;
//                }
//          }

            // Verbindung schliessen, Dateien schliessen, Session beenden
      }
```

1.7.4.4. Aufgaben

1. Starten Sie mit dem Programmrahmen und kreieren Sie ein `Properties` Objekt.
2. Kreieren Sie ein `Session` Objekt mit den `Properties`.
3. Definieren Sie ein `Store` Objekt für Ihr email Protokoll, entweder `pop3` oder `imap`.
4. Bauen Sie eine Verbindung zum Mail Server auf, mit passendem Benutzernamen und Passwort (sie können diese als Parameter eingeben).
5. Definieren Sie den Folder, aus dem Sie lesen möchten , wahrscheinlich `INBOX`.
6. Öffnen Sie den Folder read-only.
7. Bestimmen Sie eine Liste der Meldungen in Ihrem Folder. Speichern Sie die Mailing Liste in einem Array `message`.
8. Zeigen Sie für jede Meldung die Felder 'from' und 'subject' an.
9. Zeigen Sie jene Meldungen an, welche der Benutzer auswählt.
10. Schliessen Sie die Verbindung zum Folder und zum Store..
11. Übersetzen Sie das Programm und starten Sie es. Testen Sie das Programm!

1.7.4.5. Lösungshinweise

Zu Aufgabe 1

1. Starten Sie mit dem Programmrahmen und kreieren Sie ein `Properties` Objekt.

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

Zu Aufgabe 2

2. Kreieren Sie ein `Session` Objekt mit den `Properties`.

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

Zu Aufgabe 3

3. Definieren Sie ein `Store` Objekt für Ihr email Protokoll, entweder `pop3` oder `imap`.

```
Store store = session.getStore("pop3");
```

Zu Aufgabe 4

4. Bauen Sie eine Verbindung zum Mail Server auf, mit passendem Benutzernamen und Passwort (sie können diese als Parameter eingeben).

```
store.connect(host, username, password);
```

Zu Aufgabe 5

5. Definieren Sie den Folder, aus dem Sie lesen möchten , wahrscheinlich `INBOX`.

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

Zu Aufgabe 6

6. Öffnen Sie den Folder read-only.

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

Zu Aufgabe 7
7. Bestimmen Sie eine Liste der Meldungen in Ihrem Folder. Speichern Sie die Mailing Liste in einem Array `message`.

```
Message message[] = folder.getMessages();
```

Zu Aufgabe 8
8. Zeigen Sie für jede Meldung die Felder 'from' und 'subject' an.

```
System.out.println(i + ": " + message[i].getFrom()[0]
  + "\t" + message[i].getSubject());
```

Zu Aufgabe 9
9. Zeigen Sie jene Meldungen an, welche der Benutzer auswählt.

```
System.out.println(message[i].getContent());
```

Zu Aufgabe 10
10. Schliessen Sie die Verbindung zum Folder und zum Store..

```
folder.close(false);
store.close();
```

Zu Aufgabe 11
11. Übersetzen Sie das Programm und starten Sie es. Testen Sie das Programm!

1.7.4.6. Musterlösung

```
package praxis;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.Properties;

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

public class Lesen {
      public static void main(String args[]) throws Exception {
            /*
              String host = args[0];
              String username = args[1];
              String password = args[2];
            */
            String setup[] =
                  { "mail.joller-voss.ch", "UID", "PWD" };
            // 1) Properties bestimmen oder setzen
            Properties props = new Properties();
            String host = setup[0];
            String username = setup[1];
            String password = setup[2];

            // 1.a) System Properties bestimmen (Argumente)
            // Properties props = System.getProperties();

            // 2) Mailbox Server festlegen
            props.put("mail.pop3.host", host);

            // 3) Session kreieren
            Session session = Session.getDefaultInstance(props, null);
            session.setDebug(true);
```

```
            // 4) Mailbox festlegen (store)
            Store store = session.getStore("pop3");

            // 4.1) Verbindung zur Mailbox aufbauen
            store.connect(host, username, password);

            // 4.2) Folder / Mailbox bestimmen
            Folder folder = store.getFolder("INBOX");

            // 4.3) Mailbox read-only (lesend) öffnen
            folder.open(Folder.READ_ONLY);

            BufferedReader reader =
                  new BufferedReader(new InputStreamReader(System.in));

            // 5) Directory bestimmen
            javax.mail.Message[] message = folder.getMessages();

            for (int i = 0, n = message.length; i < n; i++) {

                  // 6) Inhalt
                  // 6.1) von Subject und From Feld anzeigen
                  System.out.println(
                        i
                              + ": "
                              + message[i].getFrom()[0]
                              + "\t"
                              + message[i].getSubject());

                  System.out.println(
                        "Moechten Sie die Nachricht lesen?
                        [JA zum Lesen/QUIT falls nicht]");
                  String line = reader.readLine();
                  if ("JA".equals(line)) {

                        // 6.2) Message Inhalt anzeigen
                        System.out.println(message[i].getContent());

                  } else if ("QUIT".equals(line)) {
                        break;
                  }
            }
            folder.close(false);
            store.close();
      }
}
```

1.7.4.7. Demonstration

Hier ein Beispielaufruf, mit allgemeinen Servernamen, Benutzernamen und Passwort.

Schema:

```
java Lesen POP-Server username password
```

Beispiel:

java GetMessage

```
0: president@whitehouse.gov  Thanks.
DEBUG: setDebug: JavaMail version 1.3.1
```

```
DEBUG: getProvider() returning
javax.mail.Provider[STORE,pop3,com.sun.mail.pop3.POP3Store,Sun Microsy
stems, Inc]
DEBUG POP3: connecting to host "mail.joller-voss.ch", port 110
S: +OK POP3 reo0022.wsbox.ch v2003.83rh server ready
C: USER UID
S: +OK User name accepted, password please
C: PASS PWD
S: +OK Mailbox open, 0 messages
C: STAT
S: +OK 0 0
C: QUIT
S: +OK Sayonara
```

1.7.5. Löschen von Messages und Flags

Um Meldungen zu löschen, müssen Sie mit Flags arbeiten. Jeder Meldung sind Flags zugeordnet, je nach dem Zustand der Meldung. Einige der Zustände sind systemgesteuert, andere jedoch benutzerzugänglich.. Folgende Flags sind bereits im voraus definiert:

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

Allerdings heisst es noch lange nicht, dass diese Flags auch überall / in jedem Mail System vorhanden sein müssen.

- Das POP Protokoll unterstützt beispielsweise gemäss RFC lediglich das *Löschen* von Meldungen.
- Das Prüfen, ob neue Meldungen vorliegen, wird gemäss RFC nicht verlangt. Aber die meisten Mail Clients sind froh, wenn es diese Funktion gibt.
- POP3 Folder unterstützen keine Flags.

Sie können mit `folder.getPermanentFlags()` herausfinden, welche Flags unterstützt werden.Der Folder muss aber das IMAP Protokoll unterstützen, weil POP3 Folder keine Flags unterstützen.

Falls Sie eine Meldung löschen möchten:

1) setzen Sie als erstes das `DELETED` Flag:
   ```
   message.setFlag(Flags.Flag.DELETED, true);
   ```

2) öffnen Sie den Folder im `READ_WRITE` Modus:
   ```
   folder.open(Folder.READ_WRITE);:
   ```

3) Nun können Sie ihre Nachrichten bearbeiten.

4) Beim Schliessen des Folders müssen Sie den '`expurge`' (purge on exit) Wert auf `true` setzen:
   ```
   folder.close(true);
   ```

Die `expurge()` Methode der Klasse `Folder` funktioniert nicht bei jedem Provider (POP, IMAP Klassenlibrary).

Neben dem Setzen der Flags mit der `setFlag()` Methode können Sie die Flags auch abfragen:

- `isSet()` oder
- zurücksetzen: `setFlag(false)`

1.7.5.1. Beispielprogramm

Das folgende Programm setzt voraus, dass auf dem Mail Server Folder, speziell eine Inbox vorhanden ist, welche mehrere Meldungen enthält. Für diese Meldungen werden die Flags bestimmt und ausgegeben:

```
package zusatzBeispiele;

/**
 * Title:        Setzen und Lesen von Message Flags
 * Description:  einfache Demo des Einsatzen von Flags
 */

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;

public class FlagsClient {

      public static void main(String[] args) {

            if (args.length == 0) {
                  System.err.println(
                        "Usage: java FlagsClient
                        protocol://username@host:port/foldername\n"
                   "Beispiel: pop3://fam_joller@pop.bluewin.ch/INBOX");
                  System.exit(2);
            }
            System.out.println("[FlagsClient]Start");
            URLName server = new URLName(args[0]);
            System.out.println("[FlagsClient]URL : " + args[0]);

            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);
      }

      // Flags könnte man ausgeben über
      Flags flags = folder.getPermanentFlags();
      System.out.println("----------------------------");
      System.out.println("ANSWERED Flag: "+
                  flags.contains(Flags.Flag.ANSWERED));
      System.out.println("DELETED Flag: "+
                  flags.contains(Flags.Flag.DELETED));
      System.out.println("DRAFT Flag: "+
                  flags.contains(Flags.Flag.DRAFT));
      System.out.println("FLAGGED Flag: "+
                  flags.contains(Flags.Flag.FLAGGED));
      System.out.println("RECENT Flag: "+
                  flags.contains(Flags.Flag.RECENT));
      System.out.println("SEEN Flag: "+
                  flags.contains(Flags.Flag.SEEN));
      System.out.println("USER Flag: "+
                  flags.contains(Flags.Flag.USER));
      System.out.println("-------------------------");

      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();
            }
      }
}
```

1.7.6. Authentifizierung

Wir haben bereits in den Übungen die `Authenticator` Klasse eingesetzt, mit einem einfachen GUI mit Prompts für den Benutzernamen und das Passwort. Nun wollen wir uns etwas genauer mit dieser Klasse befassen.

Bisher haben wir uns gegenüber dem Mail Host im `Store` Objekt authentifiziert:

```
// Mail Server festlegen
props.put("mail.smtp.host", host);

// Session aufbauen
Session session = Session.getDefaultInstance(props, null);
session.setDebug(true);

// Store / Mailbox Typ festlegen
Store store = session.getStore("pop3");

// Verbindung zum Host aufbauen
store.connect(host, username, password);
```

Mit dem Authenticator können wir bereits eine Stufe früher unsere Benutzerangaben verwenden, nämlich bei der Instanzierung des `Session` Klasse:

```
// setzen der Properties
Properties props = System.getProperties();
props.put("mail.pop3.host", host);

// Authenticator Objekt und Session Objekt
Authenticator auth = new PopupAuthenticator();
Session session = Session.getDefaultInstance(props, auth);

// nun stellen wir eine Verbindung her
Store store = session.getStore("pop3");
store.connect();
```

Hier eine grafische Variante, aus dem Projekt 'LesenAllerMailHeader':

```
package zusatzBeispiele;

/**
 * Title:        Mail Authenticator Beispiel (beachte: extends...)
 */

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("Benutzername: ");
	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 aus Sicherheitsgründen ein Array
		// Dieses müssen wir in eine Zeichenkette umwandeln für den
		// PasswordAuthentication()Konstruktor
		String password = new String(passwordField.getPassword());
		String username = usernameField.getText();
		// Löschen des Passwortes
		// für alle Fälle
		passwordField.setText("");
		return new PasswordAuthentication(username, password);
	}
}
```

Und hier das Anwendungsbeispiel:

```
package zusatzBeispiele;

/**
 * Title:        Lesen aller Header
 */
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;

public class LesenAllerHeaderClient {
      private static final String TARGET =
"pop3://fam_joller@pop.bluewin.ch/INBOX";
      public static void main(String[] args) {
            String mail;
            if (args.length == 0) {
                  System.err.println(
                        "Usage: java LesenAllerClient
protocol://username@host:port/foldername");
                  //return;
                  mail = TARGET;
            } else {
                  mail = args[0];
            }
            System.out.println("[LesenAllerHeaderClient]main()");
            URLName server = new URLName(mail);
            System.out.println("[LesenAllerHeaderClient.main()]url : "+
                              mail);
            try {
            System.out.println("[LesenAllerHeaderClient.main()]Session");
                  Session session =
                        Session.getDefaultInstance(
                              new Properties(),
                              new MailAuthenticator(server.getUsername()));

            // Verbindungsaufbau
            System.out.println("[LesenAllerHeaderClient.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 Nachrichten
            System.out.println("[LesenAllerHeaderClient.main()]Message[]");
                  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.close()");
                    folder.close(false);
                } catch (Exception e) {
                    e.printStackTrace();
                }
                System.out.println("[LesenAllerHeaderClient.main()]exit");
                System.exit(0);
                // muss da stehen, sonst wartet das GUI auf neuen Input
        }
}
```

1.7.7. Meldungen beantworten

Die `Message` Klasse enthält auch eine `reply()` Methode:

```
abstract Message reply(boolean replyToAll)
```

Die Methode kopiert die `'From'` Adresse in das `'To'` Feld, sonst geschieht eigentlich fast nichts. Falls der Parameter `false` ist, wird die Antwort lediglich an den Absender zurück gesandt, sonst bei `true` an alle Adressen auf der Message. Auch das `'From'` Feld muss neu gesetzt werden:

```
MimeMessage reply = (MimeMessage)message.reply(false);
reply.setFrom(new InternetAddress("president@whitehouse.gov"));
reply.setText("Thanks Mr President and ... ");
Transport.send(reply);
```

Falls Sie die Meldung an eine bestimmte Adresse senden wollen (auch als Antwort!), können Sie diese Adresse mit der `setReplyTo()` Methode abgeben:

```
void setReplyTo(Address[] addresses)
```

1.7.8. Übung - Beantworten von Meldungen

In dieser Übung schreiben wir ein Programm, welches eine Antwort auf eine Message kreiert und die alte Textmeldung hinten anhängt.

1.7.8.1. Voraussetzungen

Sie haben die Übungsaufgabe 'Lesen von Nachrichten' durchgearbeitet.

1.7.8.2. Rahmenprogramm

Das folgende Programm gilt als Basis für diese Übung:

```
package praxisvorgabe;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.Properties;

import javax.mail.Folder;
```

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

public class Reply {
	private static final String
		SMTP_HOST="smtp.meinprovider.com",
		POP3_HOST="pop3.meinspam.com",
		USER="uid",
		PWD="*****",
		REPLY_FROM="user@host.com";
	public static void main(String args[]) throws Exception {
		String pop3Host, smtpHost, username, password, from;
		if (args.length<4) {
			smtpHost = SMTP_HOST;
			pop3Host = POP3_HOST;
			username = USER;
			password = PWD;
			from = REPLY_FROM;
		} else {
			smtpHost = args[0];
			pop3Host = args[1];
			username = args[2];
			password = args[3];
			from = args[4];
		}

		// leere Properties
		Properties props = System.getProperties();
		props.put("mail.smtp.host", smtpHost);

		// Bestimmen der Session
		Session session = Session.getDefaultInstance(props, null);

		// und des Stores
		Store store = session.getStore("pop3");
		store.connect(pop3Host, username, password);

		// und des Folders
		Folder folder = store.getFolder("INBOX");
		folder.open(Folder.READ_ONLY);

		BufferedReader reader =
			new BufferedReader(new InputStreamReader(System.in));

		// Message Verzeichnis bestimmen
		Message message[] = folder.getMessages();
		for (int i = 0, n = message.length; i < n; i++) {
			System.out.println(
				i
					+ ": "
					+ message[i].getFrom()[0]
					+ "\t"
					+ message[i].getSubject());
			System.out.println(
		"Wollen Sie auf diese Meldung antworten? [JA oder/QUIT]");
			String line = reader.readLine();
			if ("JA".equals(line)) {
				// Antwort Header

				// 'From' Feld
```

```
                    // Inhalt der Antwort (der alte Text wird hineinkopiert)
                    //        Aber es wird nur Text kopiert

                          // setzen des Inhalts

                          // senden der Meldung

                    } else if ("QUIT".equals(line)) {
                          break;
                    }
              }

              // Connection schliessen
              folder.close(false); // kein purge der Meldungen
              store.close();
        }
}
```

1.7.8.3. Aufgaben

Versuchen Sie folgende Aufgaben mit dem Rahmenprogramm oben zu lösen:

1. Das Rahmenprogramm enthält bereits den Programmcode, um Meldungen zu selektieren und für eine Beantwortung auszuwählen. Prüfen Sie das Programmrahmen.
2. Im Programmteil für den Fall, dass Sie eine der Meldungen auswählen und beantworten möchten:
   ergänzen Sie das Rahmenprogramm so, dass diese Option möglich wird, also eine neue `MimeMessage` kreiert wird.
3. Setzen Sie das 'From' Feld auf Ihre eigene email Adresse.
4. Kreieren Sie einen Antworttext und ergänzen Sie Ihre Meldung durch den alten Text, sofern dieser wirklich Text war.
   Schreiben Sie vor jede Textzeile des alten Textes ein '>', damit der neue Text eindeutig erkennbar wird.
5. Setzen Sie nun den Text als Message Inhalt (Content).
6. Schreiben Sie Programmcode zum Senden der Meldung.
7. Übersetzen Sie Ihr Programm und testen Sie es. Als Programmparameter müssen Sie den Mailhost, Benutzernamen und Passwort, sowie die spezielle 'From' Adresse angeben.
8. Überprüfen Sie das Ergebnis mit Ihrem Mail Programm.

1.7.8.4. Hilfestellung

1. Das Rahmenprogramm enthält bereits den Programmcode, um Meldungen zu selektieren und für eine Beantwortung auszuwählen. Prüfen Sie das Programmrahmen.
   Das Programm sollte fehlerfrei sein. Sie können es eventuell durch ein Package ergänzen.
2. Programmteil für den Fall, dass Sie eine der Meldungen auswählen und beantworten möchten:
   ergänzen Sie das Rahmenprogramm so, dass diese Option möglich wird, also eine neue `MimeMessage` kreiert wird.
   `MimeMessage reply = (MimeMessage)message[i].reply(false);`
3. Setzen Sie das 'From' Feld auf Ihre eigene email Adresse.
   Sie finden den vorbereiteten Programmcode im Rahmenprogramm oben.

4. Kreieren Sie einen Antworttext und ergänzen Sie Ihre Meldung durch den alten Text, sofern dieser wirklich Text war.
   Schreiben Sie vor jede Textzeile des alten Textes ein '>', damit der neue Text eindeutig erkennbar wird.
   Sie können den MIME Typus der Meldung mit der Methode `mimeMessage.isMimeType("text/plain")` *überprüfen.*
5. Setzen Sie nun den Text als Message Inhalt (Content).
6. Schreiben Sie Programmcode zum Senden der Meldung.
7. Übersetzen Sie Ihr Programm und testen Sie es. Als Programmparameter müssen Sie den Mailhost, Benutzernamen und Passwort, sowie die spezielle 'From' Adresse angeben.
   `java Reply POP.Server SMTP.Server username password` from@address
   Beispiel : `%JAVA_HOME%\bin\java -classpath .;.. Reply pop3.swissonline.ch josef.m.joller ************ user@host.com`

8. Überprüfen Sie das Ergebnis mit Ihrem Mail Programm.

1.7.8.5. Musterlösung

Die Musterlösung ergibt sich durch Mergen des Rahmenprogramms mit den obigen Lösungshinweisen und ein paar Zeilen von Ihnen:

```
package praxis;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.StringReader;
import java.util.Properties;

import javax.mail.Folder;
import javax.mail.Message;
import javax.mail.Session;
import javax.mail.Store;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
/**
 * Title:        Beantworten einer Mail Message
 * Description:  Beispiel für den Einsatz der reply() Methode
 */

public class Reply {
      private static final String
            SMTP_HOST="smtp.host.com",
            POP3_HOST="pop3.host.com",
            USER="user",
            PWD="*******",
            REPLY_FROM="user@host.com";
      public static void main(String args[]) throws Exception {
            String pop3Host, smtpHost, username, password, from;
            if (args.length<4) {
                  smtpHost = SMTP_HOST;
                  pop3Host = POP3_HOST;
                  username = USER;
                  password = PWD;
                  from = REPLY_FROM;
            } else {
                  smtpHost = args[0];
                  pop3Host = args[1];
                  username = args[2];
```

```
            password = args[3];
            from = args[4];
      }

      // leere Properties : Zielhost (der Antwort)
      System.out.println("[Reply]Setzen der Properties");
      Properties props = System.getProperties();
      props.put("mail.smtp.host", smtpHost);

      // Bestimmen der Session
      System.out.println("[Reply]Session  instanzieren");
      Session session = Session.getDefaultInstance(props, null);
      session.setDebug(true);

      // und des Stores (Host mit den Mails: POP3)
      System.out.println("[Reply]Store  instanzieren");
      Store store = session.getStore("pop3");
      store.connect(pop3Host, username, password);

      // und des Folders
      System.out.println("[Reply]Folder  instanzieren");
      Folder folder = store.getFolder("INBOX");
      folder.open(Folder.READ_ONLY);

      BufferedReader reader =
            new BufferedReader(new InputStreamReader(System.in));

      // Message Verzeichnis bestimmen
      System.out.println("[Reply]Message Verzeichnis");
      Message message[] = folder.getMessages();
      if (message.length < 1)
      System.out.println("[Reply]\tMessage Verzeichnis ist leer");
      for (int i = 0, n = message.length; i < n; i++) {
            System.out.println(
                  i
                        + ": "
                        + message[i].getFrom()[0]
                        + "\t"
                        + message[i].getSubject());

            System.out.println(
      "Wollen Sie auf diese Meldung antworten? [JA oder/QUIT]");
            String line = reader.readLine();
            if ("JA".equals(line)) {

                  // Antwort Message Objekt
                  MimeMessage reply = (MimeMessage)
                        message[i].reply(false);
                  // 'From' Feld
                  reply.setFrom(new InternetAddress(from));

            // Inhalt der Antwort (der alte Text wird hineinkopiert)
            //        Aber es wird nur Text kopiert
                  MimeMessage orig = (MimeMessage) message[i];
            StringBuffer buffer = new StringBuffer("Danke\n\n");
                  if (orig.isMimeType("text/plain")) {
                        String content = (String) orig.getContent();
            StringReader contentReader = new StringReader(content);
            BufferedReader br = new BufferedReader(contentReader);
            String contentLine;
            while ((contentLine = br.readLine()) != null) {
                              buffer.append("> ");
```

```
                                    buffer.append(contentLine);
                                    buffer.append("\r\n");
                              }
                        }

                        // setzen des Inhalts
                        reply.setText(buffer.toString());

                        // senden der Meldung
                        Transport.send(reply);

                  } else if ("QUIT".equals(line)) {
                        break;
                  }
            }

            // Connection schliessen
            System.out.println("[Reply]Folder schliessen");
            folder.close(false); // kein purge der Meldungen
            System.out.println("[Reply]Store schliessen");
            store.close();
            System.out.println("[main]ENDE");
      }
```

1.7.8.6. }Demonstration

Je nach Provider werden Sie Probleme bekommen, falls Sie die Mails nicht an sich schreiben und auch sich selbst antworten.

```
[Reply]Setzen der Properties
[Reply]Session  instanzieren
DEBUG: setDebug: JavaMail version 1.3.1
[Reply]Store  instanzieren
DEBUG: getProvider() returning
javax.mail.Provider[STORE,pop3,com.sun.mail.pop3.POP3Store,Sun Microsy
stems, Inc]
DEBUG POP3: connecting to host "pop.swissonline.ch", port 110
S: +OK Hello there.
C: USER josef.m.joller
S: +OK Password required.
C: PASS ??????????
S: +OK logged in.
[Reply]Folder  instanzieren
C: STAT
S: +OK 4 3920
[Reply]Message Verzeichnis
C: TOP 1 0
S: +OK headers follow.
Return-Path: <absender@joller-voss.ch>
Received: from smtp.hispeed.ch (smtp-03-e0.tornado.cablecom.ch
[10.11.135.3])
      by mx.hispeed.ch (8.12.6/8.12.6/tornado-1.0) with ESMTP id
i27Ao47p004706
      for <josef.m.joller@swissonline.ch>; Sun, 7 Mar 2004 11:50:04 +0100
X-Sending-Host: smtp-03-e0.tornado.cablecom.ch
X-Sending-IP: 10.11.135.3
Received: from NINFPDW11 (217-162-84-44.dclient.hispeed.ch [217.162.84.44])
      by smtp.hispeed.ch (8.12.6/8.12.6/tornado-1.0) with ESMTP id
i27Ao3DG006841
      for <josef.m.joller@swissonline.ch>; Sun, 7 Mar 2004 11:50:03 +0100
Date: Sun, 7 Mar 2004 11:50:03 +0100
Message-ID: <27994366.1078656601409.JavaMail.jjoller@NINFPDW11>
From: absender@joller-voss.ch
```

```
To: josef.m.joller@swissonline.ch
Subject: Hallo aus JavaMail
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
X-Virus-Scanned: ClamAV version 'clamd / ClamAV version 0.65', clamav-
milter version '0.60p'

0: absender@joller-voss.ch    Hallo aus JavaMail
Wollen Sie auf diese Meldung antworten? [JA oder/QUIT]
JA
C: RETR 1
S: +OK 980 octets follow.
Return-Path: <absender@joller-voss.ch>
Received: from smtp.hispeed.ch (smtp-03-e0.tornado.cablecom.ch
[10.11.135.3])
      by mx.hispeed.ch (8.12.6/8.12.6/tornado-1.0) with ESMTP id
i27Ao47p004706
      for <josef.m.joller@swissonline.ch>; Sun, 7 Mar 2004 11:50:04 +0100
X-Sending-Host: smtp-03-e0.tornado.cablecom.ch
X-Sending-IP: 10.11.135.3
Received: from NINFPDW11 (217-162-84-44.dclient.hispeed.ch [217.162.84.44])
      by smtp.hispeed.ch (8.12.6/8.12.6/tornado-1.0) with ESMTP id
i27Ao3DG006841
      for <josef.m.joller@swissonline.ch>; Sun, 7 Mar 2004 11:50:03 +0100
Date: Sun, 7 Mar 2004 11:50:03 +0100
Message-ID: <27994366.1078656601409.JavaMail.jjoller@NINFPDW11>
From: absender@joller-voss.ch
To: josef.m.joller@swissonline.ch
Subject: Hallo aus JavaMail
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
X-Virus-Scanned: ClamAV version 'clamd / ClamAV version 0.65', clamav-
milter version '0.60p'
…..
354 Enter mail, end with "." on a line by itself
Message-ID: <2614099.1078656710837.JavaMail.jjoller@NINFPDW11>
From: josef.m.joller@swissonline.ch
To: absender@joller-voss.ch
Subject: Re: Hallo aus JavaMail
In-Reply-To: <27994366.1078656620817.JavaMail.jjoller@NINFPDW11>
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit

Danke

> Willkommen bei JavaMail

.
…..
250 2.0.0 i27ApqCQ001828 Message accepted for delivery
QUIT
[Reply]Folder schliessen
C: QUIT
S: +OK Bye-bye.
[Reply]Store schliessen
[main]ENDE
```

1.7.9. Meldungen weiterleiten

Das Weiterleiten von Meldungen ist etwas komplexer. Es ist Ihre Aufgabe, eine zusammengesetzte Meldung aufzubauen und zu versenden. Jeder Teil der Meldung wird als Part bezeichnet, mehrere Parts zusammen machen die gesamte Meldung aus.

Jeder Teil der Message ist ein sogenannter `BodyPart`, genauer ein `MimeBodyPart` , falls Sie mit MIME Meldungen arbeiten. Die verschiedenen Bestandteile werden in einem Container, dem `Multipart` , speziell einem `MimeMultipart` zusammengefügt.

Um beispielsweise eine Meldung weiterzuleiten, müssen Sie

1) eine Meldung kreieren und
2) in einem zweiten Teil die Meldung, welche Sie weiterleiten wollen.
3) Beide sind eine Multipart Message

Das Kopieren einer Meldung in eine neue Meldung geschieht mit einem `DataHandler`. Diese wurden ursprünglich für die JavaBeans (Activation Framework) entwickelt:

```
// Kreieren der Meldung
Message forward = new MimeMessage(session);

// Header
forward.setSubject("Fwd: " + message.getSubject());
forward.setFrom(new InternetAddress(from));
forward.addRecipient(Message.RecipientType.TO,
  new InternetAddress(to));

// Message Body
BodyPart messageBodyPart = new MimeBodyPart();
messageBodyPart.setText(
  "Hier steht die ursprüngliche Meldung:\n\n");

// Im Multipart Objekt werden die einzelnen Message Teile
// zusammengefasst
Multipart multipart = new MimeMultipart();
multipart.addBodyPart(messageBodyPart);

// Kreieren und ergänzen der weiterzuleitenden Teile
messageBodyPart = new MimeBodyPart();
messageBodyPart.setDataHandler(message.getDataHandler());

// ... und hinzufügen zum Multipart Body
multipart.addBodyPart(messageBodyPart);

// Multipart Body wird der Meldung zugewiesen
forward.setContent(multipart);

// ... und versandt
Transport.send(forward);
```

1.7.10. Mit Anhängen arbeiten

Anhänge sind Ressourcen, welche einer Mail Message zugeordnet sind und in der Regel ausserhalb der Textmeldung gehalten werden. Beispiele sind:
PDF Dateien, ein Word Dokument, ein Excel Dokument, ... ein Bild,...

Auch in Java kann man Dateien einer Meldung anhängen und versenden.

1.7.10.1. Senden von Anhängen

Das Senden von Anhängen / Attachments ist sehr ähnlich, wie das Beantworten einer Meldung. Als erstes bauen Sie die einzelnen Bausteine, dann verbinden Sie diese zu einer Meldung und versenden sie.

Dazu steht Ihnen ein sogenannter `DataHandler` zur Verfügung. Je nach der Quelle des Attachments verwenden Sie unterschiedliche Klassen, zur Beschreibung des Attachments:
- falls Sie eine Datei anhängen wollen, benötigen Sie eine `FileDataSource`.
- falls Sie eine URL anhängen wollen, benötigen Sie `URLDataSource`.

Nachdem Sie Ihre Datenquelle beschrieben haben, müssen Sie noch einen `DataHandler` konstruieren und an dessen Konstruktor die Datenquelle übergeben:

```
DataSource source = new FileDataSource(filename);
```

Schliesslich wird der Anhang mit der Methode `setDataHandler()` der Meldung angehängt:

```
messageBodyPart.setDataHandler(new DataHandler(source));
```

```
// Definition der Message
Message message = new MimeMessage(session);
message.setFrom(new InternetAddress(from));
message.addRecipient(Message.RecipientType.TO,
  new InternetAddress(to));
message.setSubject("Beispiel eines JavaMails mit Attachment");

// Kreieren der message Parts
BodyPart messageBodyPart = new MimeBodyPart();

// Angabe der message
messageBodyPart.setText("Wie geht's?");

Multipart multipart = new MimeMultipart();
multipart.addBodyPart(messageBodyPart);

// Attachment
messageBodyPart = new MimeBodyPart();
DataSource source = new FileDataSource(filename);
messageBodyPart.setDataHandler(new DataHandler(source));
messageBodyPart.setFileName(filename);
multipart.addBodyPart(messageBodyPart);

// Zusammensetzen der Meldung
message.setContent(multipart);

// Senden der Meldung
Transport.send(message);
```

1.7.11. Übung - Senden von Attachments

In dieser Übung kreieren wir ein programm, mit dem man Meldungen mit Attachments versenden kann.

1.7.11.1. Voraussetzungen

Sie haben die erste Übung (Senden einer Meldung) erfolgreich durchgearbeitet.

1.7.11.2. Rahmenprogramm

Das folgende Programm liefert Ihnen einen Rahmen für Ihre eigene Lösung:

```
package praxisvorgabe;

import java.util.Properties;
import javax.mail.Session;

public class Attach {
	private static final String
		SMTP_HOST="smtp.host.ch",
		FROM="user@host.ch",
		TO="benutzer@host.ch",
		FILENAME="Glider.bmp";
	public static void main(String args[]) throws Exception {
		String smtpHost, from, to, filename;
		if (args.length<4) {
			smtpHost = SMTP_HOST;
			from = FROM;
			to = TO;
			filename = FILENAME;
		} else {
			smtpHost = args[0];
			from = args[1];
			to = args[2];
			filename = args[3];
		}
		// Bestimmen / Setzen der System Eigenschaften
		Properties props = System.getProperties();
		// Angaben zum Mail Server
		props.put("mail.smtp.host", smtpHost);
		// bestimmen der Session / Instanzieren der Session Klasse
		Session session = Session.getInstance(props, null);
		// Definieren der Message
		// Kreieren der message Bestandteile
		// Eingabe der Meldung
		// Kreieren der Multipart Message
		// Part 1 hinzufügen
		// Part 2 Attachment
		// kreieren eines Multiparts
		// bestimmen der Attachments
		// setzen des Data Handlers auf das Attachment
		// setzen des Dateinamens
		// Teil zwei hinzufügen
		// Teile zu einer Message zusammenfügen
		// senden der Message
	}
}
```

1.7.11.3. Aufgaben

1. Überprüfen Sie das Rahmenprogramm. Es enthält alles zur Initialisierung einer Mail Session.
2. Kreieren Sie eine Meldung zur Session, mit Header, From, To und Subject.
3. Kreieren Sie einen `BodyPart` für die Hauptmeldung und geben Sie einen Text ein.
4. Kreieren Sie ein `Multipart` um die Message mit dem Attachment zu kombinieren und fügen Sie die Message dem Multipart hinzu.
5. Kreieren Sie einen zweiten `BodyPart` für das Attachment.
6. Bestimmen Sie das Attachment als `DataSource`.
7. Setzen Sie den `DataHandler` für den Message Teil der Data Source.
8. Fügen Sie den zweiten Teil der Message zum Multipart.
9. Setzen Sie den Inhalt der Message mit dem Multipart gleich.
10. Senden Sie die Meldung.
11. Übersetzen und starten Sie das Programm.
12. Überprüfen Sie das Ergebnis.

1.7.11.4. Hilfestellungen

Zu Aufgabe 3:

```
BodyPart messageBodyPart = new MimeBodyPart();
messageBodyPart.setText("Here's the file");
```

Zu Aufgabe 4:

```
Multipart multipart = new MimeMultipart();
multipart.addBodyPart(messageBodyPart);
```

Zu Aufgabe 6:

```
DataSource source = new FileDataSource(filename);
```

Zu Aufgabe 7:

```
messageBodyPart.setDataHandler(new DataHandler(source));
messageBodyPart.setFileName(filename);
```

Zu Aufgabe 9

```
message.setContent(multipart);
```

1.7.11.5. Musterlösung

Hier die Musterlösung, basierend auf dem Rahmenprogramm:

```
package praxis;

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

/**
 * Title:        Attachments in JavaMail
 * Description:  Beispiel für die Behandlung von Attachments in JavaMail
 */

public class Attach {
      private static final boolean DEBUG = true;
      public static void main(String args[]) throws Exception {
            String host, from, to, file;
            if (args.length < 4) {
                  host = "smtp.swissonline.ch";
                  from = "joller@joller-voss.ch";
                  to = "joller@joller-voss.ch";
                  file = "Glider.bmp";
            } else {
                  host = args[0];
                  from = args[1];
                  to = args[2];
                  file = args[3];
            }
            if (DEBUG) {

      System.out.println("====================================");
                  System.out.println("[main]Host   " + host);
                  System.out.println("[main]from   " + from);
                  System.out.println("[main]to     " + to);
                  System.out.println("[main]File   " + file);

      System.out.println("====================================");
            }

            // Bestimmen / Setzen der System Eigenschaften
            if (DEBUG) System.out.println("[main]Properties setzen");
            Properties props = System.getProperties();

            // Angaben zum Mail Server
            props.put("mail.smtp.host", host);

            // bestimmen der Session / Instanzieren der Session Klasse
            Session session = Session.getInstance(props, null);
            session.setDebug(false);

            // Definieren der Message
            Message message = new MimeMessage(session);
            message.setFrom(new InternetAddress(from));
            message.addRecipient(Message.RecipientType.TO,
                                 new InternetAddress(to));
            String line="Hallo JavaMail Attachment";
            if (DEBUG) System.out.println("[main]Subject:"+line);
            message.setSubject(line);
```

```
            // Kreieren der Message Bestandteile
            BodyPart messageBodyPart = new MimeBodyPart();

            // Eingabe der Meldung
            line="Auch mir geht's gut!";
            messageBodyPart.setText("Auch mir geht's gut!");
            if (DEBUG) System.out.println("[main]Body:"+line);

            // Kreieren der Multipart Message
            Multipart multipart = new MimeMultipart();

            // Part 1 hinzufügen
            multipart.addBodyPart(messageBodyPart);

            // Part 2 (Attachment) hinzufügen
            if (DEBUG) System.out.println("[main]Body:Attachment");

            // kreieren eines Multiparts
            messageBodyPart = new MimeBodyPart();

            // bestimmen der Attachments
            DataSource source = new FileDataSource(file);

            // setzen des Data Handlers auf das Attachment
            messageBodyPart.setDataHandler(new DataHandler(source));

            // setzen des Dateinamens
            messageBodyPart.setFileName(file);

            // Teil zwei hinzufügen
            multipart.addBodyPart(messageBodyPart);

            // Teile zu einer Message zusammenfügen
            message.setContent(multipart);

            // senden der Message
            Transport.send(message);
            if (DEBUG) System.out.println("[main]Ende");
        }
}
```

1.7.11.6. Demonstration

Das Programm muss mit folgendem Befehl gestartet werden:

```
java Attach SMTP.Server from@address to@address filename
```

Beispiel:

```
java Attach pop3.swissonline.ch Henry.Ford@Ford.com Edison@GEC.com x.txt
```

```
====================================
[main]Host  smtp.swissonline.ch
[main]from  joller@joller-voss.ch
[main]to    joller@joller-voss.ch
[main]File  Glider.bmp
====================================
[main]Properties setzen
[main]Subject:Hallo JavaMail Attachment
[main]Body:Auch mir geht's gut!
[main]Body:Attachment
[main]Ende
```

1.7.12. Anhänge aus einer Meldung lesen

Das Herauslesen eines Anhanges aus einer Meldung ist komplexer als das Versenden. MIME kennt den Begriff des Attachments nicht. Der Inhalt der Message ist ein Multipart Objekt. Daher muss man jeden Part, jeden Teil der Message einzeln behandeln.

Falls der Anhang gekennzeichnet wurde, erkennen Sie ihn als
`Part.ATTACHMENT`

aus

```
part.getDisposition().
```

Aber ein Attachment kann auch als
`Part.INLINE`

daherkommen.

Mit der Methode `getFileName()` können Sie den Dateinamen bestimmen.
Einen Eingabestrom erhalten Sie mit `getInputStream()`.

```
Multipart mp = (Multipart)message.getContent();

for (int i=0, n=multipart.getCount(); i<n; i++) {
  Part part = multipart.getBodyPart(i));

  String disposition = part.getDisposition();

  if ((disposition != null) &&
      ((disposition.equals(Part.ATTACHMENT) ||
       (disposition.equals(Part.INLINE))) {
    saveFile(part.getFileName(), part.getInputStream());
  }
}
```

Die `saveFile()` Methode kreiert eine Datei `File` mit dem Dateinamen, liest die Daten und schreibt die Datei.

```
// saveFile() Auszug
File file = new File(filename);
for (int i=0; file.exists(); i++) {
  file = new File(filename+i);
}
```

Im komplexen Fall könnte ein Programmfragment folgendermassen aussehen:

```
if (disposition == null) {
  // ist es einfacher Text?
  MimeBodyPart mbp = (MimeBodyPart)part;
  if (mbp.isMimeType("text/plain")) {
    // Handle plain
  } else {
    // spezielle MIME Typen image/gif, text/html, ...
  }
...
}
```

1.7.13. Verarbeiten von HTML Meldungen

Falls Sie eine HTML basierte Meldung senden wollen, wird das Programm etwas komplexer als bei einer reinen Textmeldung.

1.7.13.1. Senden einer HTML Meldung

Im einfachsten Fall senden Sie eine HTML Seite als Meldung und überlassen es dem email System des Empfängers, alles korrekt zu interpretieren, beispielsweise eingebettete Bilder.

Mit `setContent()` können Sie den MIME Type angeben; den HTML Text können Sie als Zeichenkette definieren.

```
String htmlText = "<H1>Hello</H1>"+
                  "<img src=\"http://www.host.com/images/neustes.gif\">";
message.setContent(htmlText, "text/html"));
```

Falls Sie eine HTML in JavaMail empfangen, liefert Ihnen JavaMail keinerlei Hinweise darauf, dass es sich um HTML, nicht um reinen Text handelt. JavaMail empfängt einfach einen Byte-Stream.

Falls Sie den `JEditorPane` von Swing verwenden, liefert der Ihnen gleich eine sinnvolle Darstellung als HTML Seite:

```
if (message.getContentType().equals("text/html")) {
  String content = (String)message.getContent();
  JFrame frame = new JFrame();
  JEditorPane text = new JEditorPane("text/html", content);
  text.setEditable(false);
  JScrollPane pane = new JScrollPane(text);
  frame.getContentPane().add(pane);
  frame.setSize(300, 300);
  frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
  frame.show();
}
```

1.7.13.2. Bilder in Messages

Falls Ihre Meldung neben HTML auch Bilder enthält, wird die Angelegenheit komplexer. Die Bilder sind Attachments und müssen entsprechend behandelt werden.

Die Nachricht enthält in diesem Fall eine spezielle Content-ID (`cid` URL), wobei sich die cid auf die Content-ID im Header des Attachments bezieht.

Sonst lassen sich HTML Seiten mit Bildern ähnlich wie Meldungen mit Attachments behandeln. Allerdings müssen Sie dem MimeMultipart mitteilen, dass es Untertypen gibt: `setSubType();` zudem wird dem Konstruktor eine `Content-ID` übergeben (als URL auf die Source im Image Tag).

```
String file = „Glider.bmp“;

// Kreiere die Nachricht
Message message = new MimeMessage(session);

// Header definieren
message.setSubject("melding mit Bild");
message.setFrom(new InternetAddress(from));
message.addRecipient(Message.RecipientType.TO,
  new InternetAddress(to));

// kreieren der Message Parts
BodyPart messageBodyPart = new MimeBodyPart();
String htmlText = "<H1>Bild vom Erstflug</H1>" +
  "<img src=\"cid:hoehenflug\">";
messageBodyPart.setContent(htmlText, "text/html");

// kreieren der zusammengehörenden Teile
MimeMultipart multipart = new MimeMultipart("related");
multipart.addBodyPart(messageBodyPart);

// Bildteil
messageBodyPart = new MimeBodyPart();

// Lies das Bild und definiere die Datenquelle
DataSource fds = new FileDataSource(file);
messageBodyPart.setDataHandler(new DataHandler(fds));
messageBodyPart.setHeader("Content-ID","hoehenflug");

// dem Multipart Body hinzufügen
multipart.addBodyPart(messageBodyPart);

// Definition der Message
message.setContent(multipart);
```

1.7.14. Übung - Senden einer HTML Message mit Bildern

In dieser Übung senden Sie eine HTML Seite mit einem Bild als Attachment, aber so, dass das Bild in der HTML Seite angezeigt wird.

1.7.14.1. Voraussetzungen

Sie haben die Übung über Attachments durchgearbeitet.

1.7.14.2. Rahmenprogramm

```
package praxisvorgabe;

import java.util.Properties;
import javax.activation.DataHandler;
import javax.activation.DataSource;
import javax.activation.FileDataSource;
import javax.mail.BodyPart;
import javax.mail.Message;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeBodyPart;
import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeMultipart;
```

```
/**
 * Title:        JavaMail einer HTML Seite mit Image
 * Description:  Senden einer HTML Seite, mit Bild, als eine JavaMail
Message (Multipart Message mit Content ID für das Attachment).
 */

public class HTMLmitImageMessage {
      public static final boolean DEBUG=true;
      public static void main(String args[]) throws Exception {
            String host,from,to,file;
            if (args.length<4) {
                  host="smtp.host.ch";
                  from="user1@host1.ch";
                  to = "user2@host2.ch";
                  file="Glider.bmp";
            } else {
                  host = args[0];
                  from = args[1];
                  to = args[2];
                  file = args[3]; //(javalogo52x88.gif ist klein)
            }

            System.out.println("[main]Properties setzen");
            // Bestimmen der System Properties
            Properties props = System.getProperties();
            // Mail Server
            props.put("mail.smtp.host", host);

            // Session
            Session session = Session.getDefaultInstance(props, null);

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

            // Headers
            String line="mit Bild im Anhang (kein Virus)";
            if (DEBUG) System.out.println("[main]setSubject("+line+")");
            message.setSubject(line);
            message.setFrom(new InternetAddress(from));
            message.addRecipient(Message.RecipientType.TO,
                                 new InternetAddress(to));

            // Message Parts
            BodyPart messageBodyPart = new MimeBodyPart();

            // HTML Inhalt mit Referenz auf das Attachment (Bild)
            String htmlText = "<H1>Danke</H1>" + "<img src=\"cid:bild\">";
            // Content des Body Parts
            // Multipart definieren
            // Body Part dem Multipart hinzufügen
            // Image Part definieren
            // Lesen des Images und Zuordnung an den Part
            // Header Verbindung zum HTML
            // Dem Multipart Objekt hinzufügen
            // Multipart mit Message assoziieren
            // Message senden
      }
}
```

1.7.14.3. Aufgaben

1. Das Rahmenprogramm enthält bereits Programmcode für die (to, from, subject) Felder.
2. Kreieren eines `BodyPart` für den HTML Message Inhalt.
3. Definition einer Mini-HTML Seite mit einem Image Tag(<img src="...">). Die Image Source ist die email Message.
4. Definieren des Message Contents. content of the message part. MIME Type ist `text/html`.
5. Kreieren eines Multipart Objekts, um Inhalt der HTML Message mit dem Bild verknüpfen zu können. Hinzufügen des Contents zum Multipart Objekt.
6. Zweites `BodyPart` für das Attachment kreieren.
7. Attachment als `DataSource` definieren und `DataHandler` für die Message.
8. `Content-ID` Header passend zum HTML Text definieren.
9. MultiPart durch den zweiten Teil der Message ergänzen.
10. Senden der Message.
11. Übersetzen des Programms und senden des HTML Textes mit Bild.
12. Prüfen, ob die Meldung korrekt gesendet wurde.

1.7.14.4. Hilfestellung

Zu Aufgabe 4:

```
messageBodyPart.setContent(htmlText, "text/html");
```

Zu Aufgabe 5:

```
MimeMultipart multipart = new MimeMultipart("related");
multipart.addBodyPart(messageBodyPart);
```

Zu Aufgabe 11:

```
java HtmlmitImageMessage SMTP.Server from@address to@address filename
```

Beispiel:

```
java HTMLmitImageMessage pop3.host.ch u1@h1.com u2@h2.com image.gif
```

1.7.14.5. Musterlösung

```
package praxis;

import java.util.Properties;

import javax.activation.DataHandler;
import javax.activation.DataSource;
import javax.activation.FileDataSource;
import javax.mail.BodyPart;
import javax.mail.Message;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeBodyPart;
import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeMultipart;

/**
 * Title:        JavaMail einer HTML Seite mit Image
 * Description:  Senden einer HTML Seite, mit Bild, als eine JavaMail
Message (Multipart Message mit Content ID für das Attachment).
```

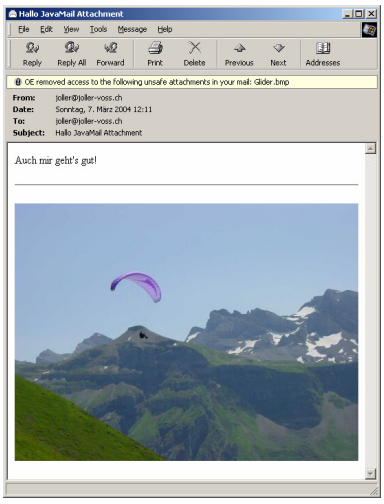
```
 */

public class HTMLmitImageMessage {
      public static final boolean DEBUG=true;
      public static void main(String args[]) throws Exception {
            String host,from,to,file;
            if (args.length<4) {
                  host="smtp.swissonline.ch";
                  from="joller@joller-voss.ch";
                  to = "joller@joller-voss.ch";
                  file="Glider.bmp";
            } else {
                  host = args[0];
                  from = args[1];
                  to = args[2];
                  file = args[3]; //javalogo52x88.gif
            }
            if (DEBUG) {

      System.out.println("====================================");
                  System.out.println("[main]Host  "+host);
                  System.out.println("[main]from  "+from);
                  System.out.println("[main]to    "+to);
                  System.out.println("[main]File  "+file);

      System.out.println("====================================");
            }

            System.out.println("[main]Properties setzen");
            // Bestimmen der System Properties
            Properties props = System.getProperties();
            // Mail Server
            props.put("mail.smtp.host", host);

            // Session
            Session session = Session.getDefaultInstance(props, null);

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

            // Headers
            String line="mit Bild im Anhang (kein Virus)";
            if (DEBUG) System.out.println("[main]setSubject("+line+")");
            message.setSubject(line);
            message.setFrom(new InternetAddress(from));
            message.addRecipient(Message.RecipientType.TO,
                        new InternetAddress(to));

            // Message Parts
            BodyPart messageBodyPart = new MimeBodyPart();

            // HTML Inhalt mit Referenz auf das Attachment (Bild)
            String htmlText = "<H1>Danke</H1>" +
                                          "<img src=\"cid:bild\">";

            // Content des Body Parts
            if (DEBUG) System.out.println("[main]setContent("+
                                                htmlText+")");
            messageBodyPart.setContent(htmlText, "text/html");

            // Multipart definieren
            MimeMultipart multipart = new MimeMultipart("related");
```

```
            // Body Part dem Multipart hinzufügen
            multipart.addBodyPart(messageBodyPart);

            // Image Part definieren
            messageBodyPart = new MimeBodyPart();

            // Lesen des Images und Zuordnung an den Part
            DataSource fds = new FileDataSource(file);
            messageBodyPart.setDataHandler(new DataHandler(fds));

            // Header Verbindung zum HTML: wie beim Header
            messageBodyPart.setHeader("Content-ID", "bild");

            // Dem Multipart Objekt hinzufügen
            multipart.addBodyPart(messageBodyPart);

            // Multipart mit Message assoziieren
            message.setContent(multipart);

            // Message senden
            if (DEBUG) System.out.println("[main]send(message)");
            Transport.send(message);
            if (DEBUG) System.out.println("[main]Ende");
        }
}
```

1.7.14.6. Demonstration

Hier ein Beispiel:

```
====================================
[main]Host   smtp.swissonline.ch
[main]from   joller@joller-voss.ch
[main]to     joller@joller-voss.ch
[main]File   Glider.bmp
====================================
[main]Properties setzen
[main]setSubject(mit Bild im Anhang (kein Virus))
[main]setContent(<H1>Danke</H1><img src="cid:bild">)
[main]send(message)
[main]Ende
```



1.7.15. Suchen mit Suchbegriffen

JavaMail enthält auch die Möglichkeit Nachrichten zu filtern. Diese Funktionen stehen im `javax.mail.search` Package

Die Suche geschieht mit Hilfe sogenannter `SearchTerms`:

```
SearchTerm st = ...;
Message[] msgs = folder.search(st);
```

Insgesamt existieren mehr als 20 Klassen, um Suchen zusammenzustellen.

- AND Terme (Klasse `AndTerm`)
- OR Terme (Klasse `OrTerm`)
- NOT t Terme (Klasse `NotTerm`)
- SENT DATE Terme (Klasse `SentDateTerm`)
- CONTENT Terme (Klasse `BodyTerm`)
- HEADER Terme (`FromTerm` / `FromStringTerm`, `RecipientTerm` / `RecipientStringTerm`, `SubjectTerm`, etc.)

Zuerst bildet man eine logische Abfrage, mit Such Termen:

```
SearchTerm st =
  new OrTerm(
    new SubjectTerm("ADV:"),
    new FromStringTerm("friend@public.com"));
```

Und nun suchen wir nach diesen Kriterien in den Messages:

```
Message[] msgs = folder.search(st);
```

1.7.15.1. Beispiel

Sun liefert mit JavaMail ein Beispiel zu Search aus:

- Sie finden das Beispiel auf dem Server (beispielsweise unter `%JAVA_MAIL%\demo\search.java`
- Im Theorieteil finden Sie weitere Beispiele.

1.7.16. Ressourcen

Falls Sie weitere Informationen zum JavaMail benötigen:

- JavaMail API Home
  `http://java.sun.com/products/javamail/`
- JavaBeans Activation Framework Home
  `http://java.sun.com/products/javabeans/glasgow/jaf.html`
- javamail-interest mailing list
  `http://mail.java.sun.com/archives/javamail-interest.html`
- Sun's JavaMail FAQ
  `http://java.sun.com/products/javamail/FAQ.html`