Primitives for Security Audits: Lessons from Jakarta Mail

Jia Hao Poh

$ whoami

- Jia Hao Poh
  - Senior Consultant @ Elttam
  - Pentesting
  - Web Application Vulnerability Research

elttam

When was the last time you dived into library implementations?

Agenda

- Background on Jakarta Mail
- Encoded Email Strings (RFC 2047)
- Primitives in Jakarta Mail
  - InternetAddress.java
  - MimeMessage.java
- Primitives in Spring Framework
  - MimeMessageHelper.java
  - InternetAddressEditor.java
  - MimeMailMessage.java and SimpleMailMessage.java
- Bonus Content: Hibernate Validator
  - @Email annotation

elttam

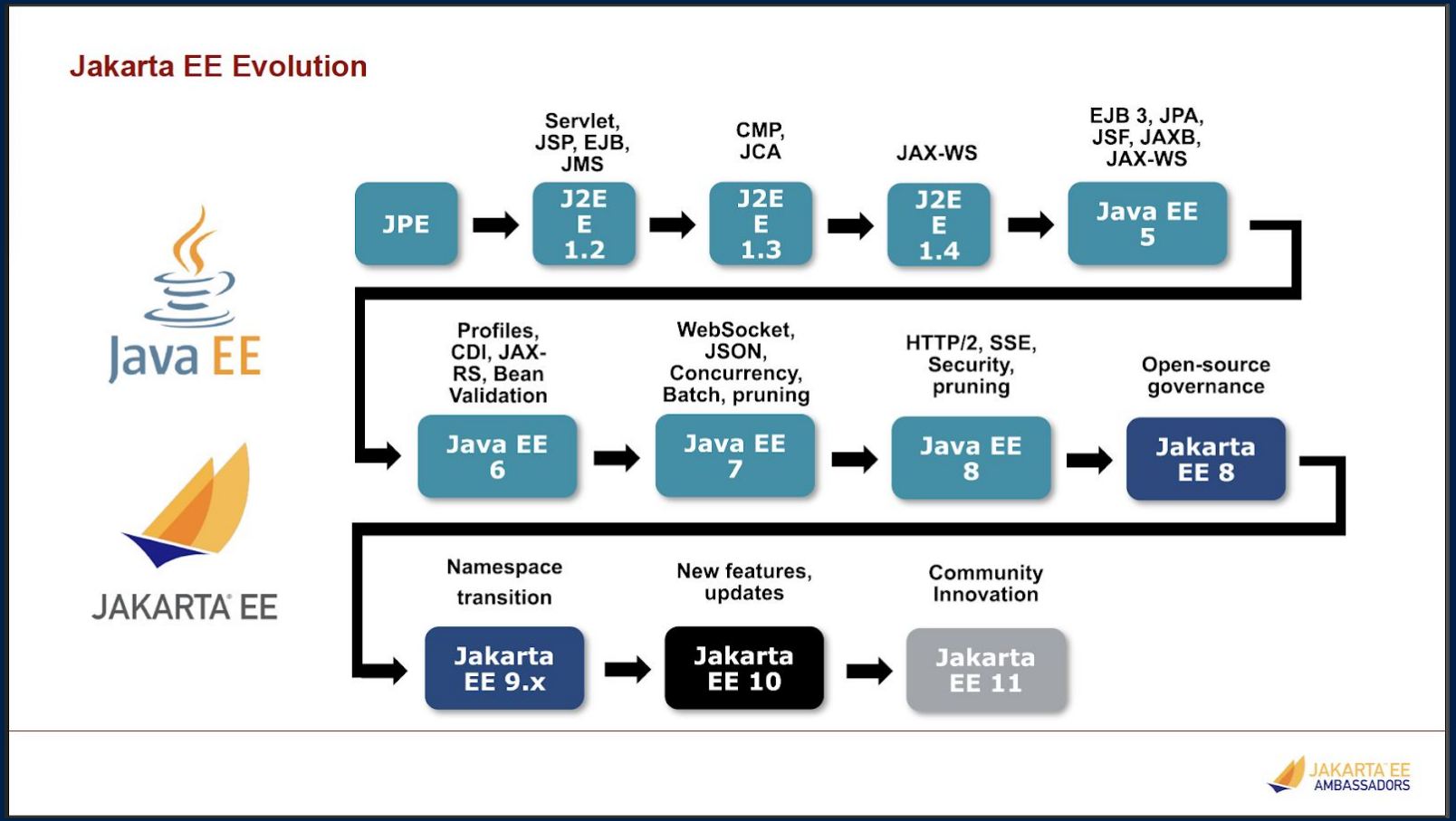
Background on Jakarta Mail

elttam

What’s in a name?

- 1999 – **Java 2 Enterprise Edition (J2EE) 1.2**
  - Specifications relating to “Enterprise” technologies
- 2006 – **Java Enterprise Edition (Java EE) 5**
  - More additions (Annotations “@”)
- 2019 – **Jakarta Enterprise Edition (Jakarta EE) 8**
  - Full compatibility with Java EE 8

elttam

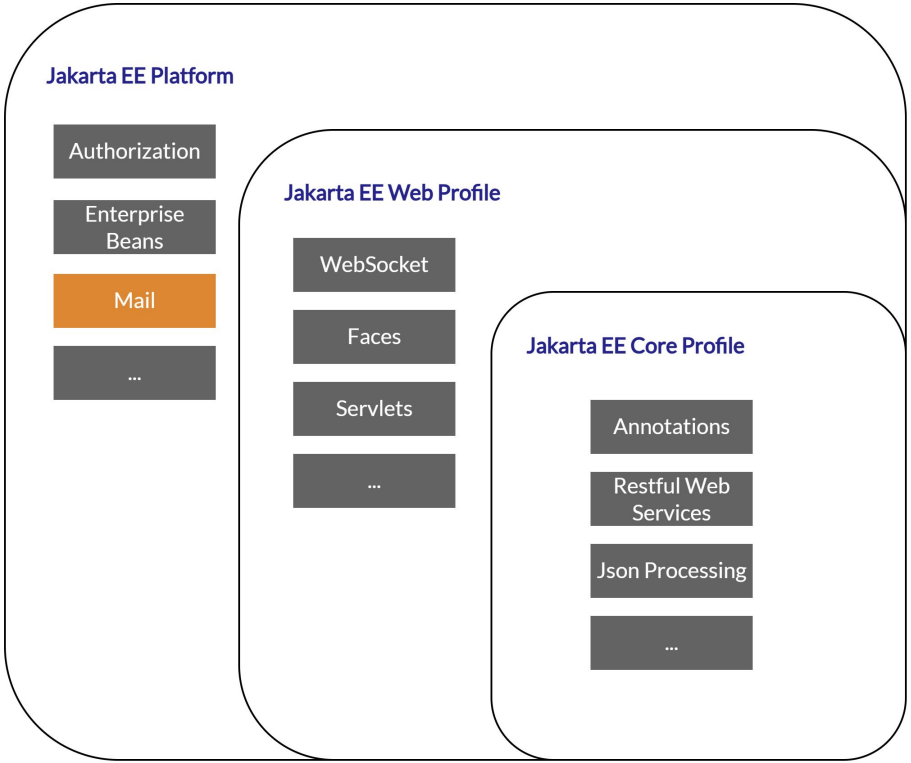


Source: https://newsroom.eclipse.org/eclipse-newsletter/2024/july/developer%E2%80%99s-guide-jakarta-ee-11

Jakarta EE Platform

Developers can pick a subset of the whole platform to be compliant with, known as **“Profiles”**.

... or even individual specifications from various aspects of an enterprise Java application, such as **Jakarta Mail**.

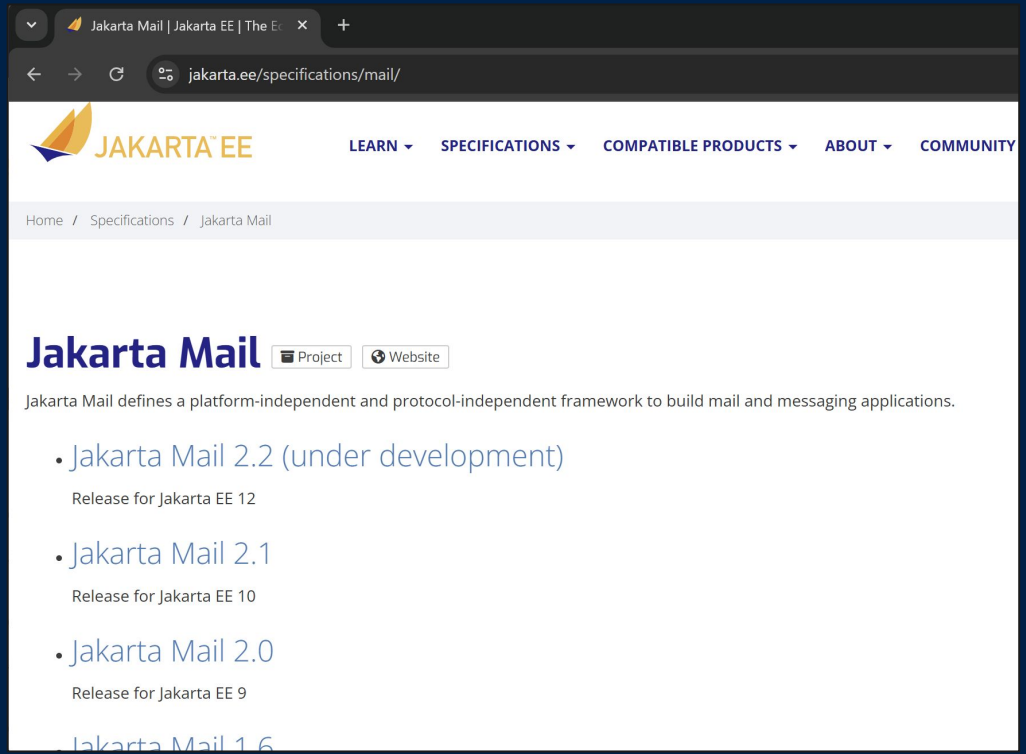


Full list of specifications can be found at: https://jakarta.ee/specifications/

elttam

Jakarta Mail

Current version is **2.1**, since **Jakarta EE 10**



elttam

Jakarta Mail

Current version is **2.1**, since **Jakarta EE 10**

Angus Mail is the only compatible implementation

Jakarta Mail | Jakarta EE | The Ec

jakarta.ee/specifications/mail/

Jakarta Mail 2.1 | Jakarta EE | Th

jakarta.ee/specifications/mail/2.1/

Details

- Jakarta Mail 2.1 Release Record
  - Jakarta EE Platform 10 Release Plan
- Jakarta Mail 2.1 Specification Document (PDF)
- Jakarta Mail 2.1 Specification Document (HTML)
- Jakarta Mail 2.1 Javadoc
- Jakarta Mail 2.1 TCK (sig, sha, pub)
  - Addresses Mail Challenge (Issue #625) Jakarta Mail 2.1.1 TCK (sig, sha, pub)
  - Allows certification on Java SE 21+ Jakarta Mail 2.1.2 TCK (sig, sha, pub)
- Maven coordinates
  - jakarta.mail:jakarta.mail-api:jar:2.1.3
- Change Log

Compatible Implementations

- Angus Mail 2.0.2

elttam

Encoded Email Strings (RFC 2047)

Please mail your concerns to:

=?utf-8?q?hello=77=6f=72=6c=64?=@example.com

elttam

Encoded Strings

=?charset?encoding?encoded-text?=

elttam

Encoded Strings

=?charset?encoding?encoded-text?=

Where:

- **=?** and **?=** are start and end anchors, **?** as separator

elttam

Encoded Strings

=?charset?encoding?encoded-text?=

Where:

- **=?** and **?=** are start and end anchors, **?** as separator
- **charset** indicates the character set of the encoded text (e.g. **UTF-8**)

elttam

Encoded Strings

=?charset?encoding?encoded-text?=

Where:

- **=?** and **?=** are start and end anchors, **?** as separator
- **charset** indicates the character set of the encoded text (e.g. **UTF-8**)
- **encoding** is either **b** (base-64) or **q** (quoted) to indicate the encoding type

elttam

Encoded Strings

=?charset?encoding?encoded-text?=

Where:

- **=?** and **?=** are start and end anchors, **?** as separator
- **charset** indicates the character set of the encoded text (e.g. **UTF-8**)
- **encoding** is either **b** (base-64) or **q** (quoted) to indicate the encoding type
- **encoded-text** being the text encoded by the chosen encoding

elttam

Encoded Strings

=?utf-8?q?hello=77=6f=72=6c=64?=@example.com

Where:

- **=? ?= –** start/end markers
- **charset – UTF-8**
- **encoding –** q **(quoted)**
- **encoded-text –** hello=77=6f=72=6c=64 **(helloworld)**

elttam

Primitives in Jakarta Mail

Default classes:

InternetAddress.java and **MimeMessage.java**

elttam

InternetAddress.java

1-argument constructor:

- Single String argument sent to **parse()**
- **parse()** checks for RFC 822 compliance
- Assigns the parsed **email address**, **personal name** and **encoded personal name** to itself

InternetAddress.java

```java
public InternetAddress(String address) throws AddressException {
    // use our address parsing utility routine to parse the string
    InternetAddress[] a = parse(address, true);
    // if we got back anything other than a single address, it's an error
    if (a.length != 1)
        throw new AddressException("Illegal address", address);

    /*
     * Now copy the contents of the single address we parsed
     * into the current object, which will be returned from the
     * constructor.
     * XXX - this sure is a round-about way of getting this done.
     */
    this.address = a[0].address;
    this.personal = a[0].personal;
    this.encodedPersonal = a[0].encodedPersonal;
}
```

elttam

InternetAddress.java

```java
/**
 * Construct an InternetAddress given the address and personal name.
 * The address is assumed to be a syntactically valid RFC822 address.
 *
 * @param address  the address in RFC822 format
 * @param personal the personal name
 * @throws UnsupportedEncodingException if the personal ...
 */
public InternetAddress(String address, String personal)
        throws UnsupportedEncodingException {
    this(address, personal, null);
}

/**
 * Construct an InternetAddress given the address and personal name.
 * The address is assumed to be a syntactically valid RFC822 address.
 *
 * @param address  the address in RFC822 format
 * @param personal the personal name
 * @param charset  the MIME charset for the name
 * @throws UnsupportedEncodingException if the personal name ...
 */
public InternetAddress(String address, String personal, String charset)
        throws UnsupportedEncodingException {
    this.address = address;
    setPersonal(personal, charset);
}
```

```
// 1 Arg
InternetAddress addr = new InternetAddress("(blah)");

Caused by: jakarta.mail.internet.AddressException: Illegal address in string ``(blah)''
        at jakarta.mail.internet.InternetAddress.<init>(InternetAddress.java:103)
        ...

// 2 Args
InternetAddress addr = new InternetAddress("(blah)", "blah")
System.out.println(addr.toString()); // blah <(blah)>

//3 Args
InternetAddress addr = new InternetAddress("(blah)", "blah", "blah");
System.out.println(addr.toString()); // blah <(blah)>
```

elttam

InternetAddress::parse()

```
String email = "<aaa@bbb.com>ccc@ddd.com";
InternetAddress address = new InternetAddress(email);

StringBuilder out = new StringBuilder();
out.append("Received Email: " + email);
out.append("\n\n=====\n");
out.append("\ngetAddress(): " + address.getAddress());
out.append("\ngetPersonal(): " + address.getPersonal());
out.append("\ntoString(): " + address.toString());

return out.toString();
```

elttam

InternetAddress::parse()

```
Received Email: <aaa@bbb.com>ccc@ddd.com

=====

getAddress(): aaa@bbb.com
getPersonal(): null
toString(): aaa@bbb.com
```

Imagine this...

- Application grants special privileges to **foo.com** domain user accounts
- Registration is not restrictive enough. Attacker registers with:

  <attacker@example.com>@foo.com
- Verification mail sent to **attacker@example.com**
- Application does a naive **lastIndexOf("@")** match and sees **foo.com**
- Profit!!! 🤑

elttam

InternetAddress::parse()

```
String email = "=?utf-8?q?hello=77=6f=72=6c=64?=@example.com";
InternetAddress address = new InternetAddress(email);
address.getAddress(); // =?utf-8?q?hello=77=6f=72=6c=64?=@example.com
```

elttam

InternetAddress::parse()

```
/*
 * RFC822 Address parser.
 *
 * XXX - This is complex enough that it ought to be a real parser,
 *       not this ad-hoc mess, and because of that, this is not perfect.
 *
 * XXX - Deal with encoded Headers too.
 */
@SuppressWarnings("fallthrough")
private static InternetAddress[] parse(String s, boolean strict, boolean parseHdr) throws AddressException {
```

elttam

InternetAddress::parse()

```
/*
 * RFC822 Address parser
 *
 * XXX - This is complex
 *       not this ad-hoc
 *
 * XXX - Deal with encod
 */
@SuppressWarnings("fallt
private static InternetA                                                    hrows AddressException {
```

```
774     private static InternetAddress[] parse(String s, boolean strict,
880                         c = s.charAt(index);
881                         switch (c) {
882                             case '\\':    // XXX - is this needed?
883                                 index++; // skip both '\' and the escaped char
884                                 break;
885                             case '"':
886                                 inquote = !inquote;
887                                 break;
888                             case '>':
889                                 if (inquote)
890                                     continue;
891                                 break outf; // out of for loop
892                             default:
893                                 break;
894                         }
895                     }
896
897                     // did we find a matching quote?
898                     if (inquote) {
899                         if (!ignoreErrors)
900                             throw new AddressException("Missing '\"'", s, index);
901                         // didn't find matching quote, try again ignoring quotes
902                         // (e.g., ``<"@foo.com>'')
903                         outq:
904                         for (index = rindex + 1; index < length; index++) {
905                             c = s.charAt(index);
906                             if (c == '\\')    // XXX - is this needed?
907                                 index++;    // skip both '\' and the escaped char
908                             else if (c == '>')
```

InternetAddress::getGroup()

group-name:[addr1, addr2 ...];

Where:

- **group-name** is just a sequence of characters
- **addr1, addr2, ...** is 0 or addresses

elttam

InternetAddress::getGroup()

group-name:[addr1, addr2 ...];

isGroup

```
public boolean isGroup()
```

Indicates whether this address is an RFC 822 group address. Note that a group address is different than the mailing list addresses supported by most mail servers. Group addresses are rarely used; see RFC 822 for details.

Returns:

true if this address represents a group

Since:

JavaMail 1.3

elttam

InternetAddress::getGroup()

```
String email = "a:ccc@ddd.com,eee@fff.com,ggg@hhh.com;"

InternetAddress address = new InternetAddress(email);
InternetAddress[] addresses = address.getGroup(false);
```

elttam

InternetAddress::getGroup()

```
Received Email: a:ccc@ddd.com,eee@fff.com,ggg@hhh.com;

=====

getAddress(): a:ccc@ddd.com,eee@fff.com,ggg@hhh.com;
getPersonal(): null
toString(): a:ccc@ddd.com,eee@fff.com,ggg@hhh.com;
```

elttam

MimeMessage.java

- Used for parsing the message envelope
  - Email headers and body

- Certain headers like **“From:”**, **“Reply-to:”** and “**Subject:**” will have its value sent to **InternetAddress.parseHeader()**

MimeMessage.java

```
private Address[] getAddressHeader(String name) throws MessagingException
{
    String s = getHeader(name, ",");
    return (s == null) ? null : InternetAddress.parseHeader(s, strict);
}
```

elttam

InternetAddress.java

```java
public static InternetAddress[] parseHeader(String addresslist, boolean strict)
throws AddressException {
    return parse(MimeUtility.unfold(addresslist), strict, true);
}
```

elttam

MimeMessage.java

MimeMessage.java

```
// Note: The MimeMessage(Session session) constructor does not invoke parse().
MimeMessage(MimeMessage source);
MimeMessage(Session session, InputStream is);
MimeMessage(Folder folder, InputStream is, int msgnum);
```

elttam

MimeMessage.java

```
From: =?UTF-8?Q?Administrator_=3Cadmin@example.com=3E?= <attacker@evil.com>
To: victim@example.com
Subject: =?UTF-8?Q?Administrator_=3Cadmin@example.com=3E?=taint
Content-Type: text/plain; charset=UTF-8

Your account needs verification.
```

elttam

MimeMessage.java

```java
InputStream is = getClass().getClassLoader().getResourceAsStream("MimeMessageTest.eml");
Session session = Session.getInstance(new Properties());

// Parse the email
MimeMessage message = new MimeMessage(session, is);

// Extract raw From header
String rawFrom = message.getHeader("From", null);

// Extract raw Subject header
String rawSubject = message.getHeader("Subject", null);

// Output
Address[] froms = message.getFrom();
InternetAddress ia = (InternetAddress) froms[0];
out.append("\nRaw From Header: " + rawFrom);
out.append("\nRaw Subject Header: " + rawSubject);
out.append("\ngetSubject(): " + message.getSubject());
out.append("\ngetPersonal(): " + ia.getPersonal());
out.append("\ngetAddress(): " + ia.getAddress());
```

MimeMessage.java

```
Raw From Header: =?UTF-8?Q?Administrator_=3Cadmin@example.com=3E?= <attacker@evil.com>
Raw Subject Header: =?UTF-8?Q?Administrator_=3Cadmin@example.com=3E?=taint
getSubject(): Administrator <admin@example.com>taint
getPersonal(): Administrator <admin@example.com>
getAddress(): attacker@evil.com
```

elttam

Imagine this (again)...

- Application accepts email envelopes as input (**.eml** files)
- Some kind of input filter is used to check the raw envelope to strip denylisted words
- Attacker uses **encoded strings** to subvert the filter
- Application calls **MimeMessage(Session, InputStream)** to parse the email
- Encoded strings gets **decoded** in the resultant MimeMessage object
- Profit!!! 🤑

elttam

MimeMessage::getRecipients(Message.RecipientType)

- Retrieves a header from the email envelope
  - To: / CC: / BCC: / **Newsgroups:**

MimeMessage.java

```java
@Override
public Address[] getRecipients(Message.RecipientType type) throws MessagingException {
    if (type == RecipientType.NEWSGROUPS) {
        String s = getHeader("Newsgroups", ",");
        return (s == null) ? null : NewsAddress.parse(s);
    } else
        return getAddressHeader(getHeaderName(type));
    }
```

elttam

MimeMessage::getRecipients(Message.RecipientType)

- Retrieves a header from the email envelope
  - To: / CC: / BCC: / **Newsgroups:**

MimeMessage.java

```java
@Override
public Address[] getRecipients(Message.RecipientType type) throws MessagingException {
    if (type == RecipientType.NEWSGROUPS) {
        String s = getHeader("Newsgroups", ",");
        return (s == null) ? null : NewsAddress.parse(s);
    } else
        return getAddressHeader(getHeaderName(type));
    }
```

elttam

MimeMessage::getRecipients(Message.RecipientType)

NewsAddress.java

```
public static NewsAddress[] parse(String newsgroups)
        throws AddressException {
    // XXX - verify format of newsgroup name?
    StringTokenizer st = new StringTokenizer(newsgroups, ",");
    List<NewsAddress> nglist = new ArrayList<>();
    while (st.hasMoreTokens()) {
        String ng = st.nextToken();
        nglist.add(new NewsAddress(ng));
    }
    return nglist.toArray(new NewsAddress[0]);
}
```

elttam

MimeMessage::getRecipients(Message.RecipientType)

NewsAddress.java

```java
public static NewsAddress[] parse(String newsgroups)
        throws AddressException {
    // XXX - verify format of newsgroup name?
    StringTokenizer st = new StringTokenizer(newsgroups, ",");
    List<NewsAddress> nglist = new ArrayList<>();
    while (st.hasMoreTokens()) {
        String ng = st.nextToken();
        nglist.add(new NewsAddress(ng));
    }
    return nglist.toArray(new NewsAddress[0]);
}
```

elttam

MimeMessage::getRecipients(Message.RecipientType)

NewsAddress.java

```java
public NewsAddress(String newsgroup) {
    this(newsgroup, null);
}

public NewsAddress(String newsgroup, String host) {
    // XXX - this method should throw an exception so we can report
    // illegal addresses, but for now just remove whitespace
    this.newsgroup = newsgroup.replaceAll("\\s+", "");
    this.host = host;
}
```

elttam

MimeMessage::getRecipients(Message.RecipientType)

```
From: Alice <alice@example.com>, Bob <bob@example.org>
Newsgroups: lorem ipsum
Sender: Mailer <no-reply@example.com>
Subject: Hello
Newsgroups: foo
Content-Type: text/plain; charset=UTF-8
Newsgroups: Bar

Hi, how are you?
```

elttam

MimeMessage::getRecipients(Message.RecipientType)

```
Raw Newsgroups Header (, delimiter): lorem ipsum,foo,Bar
getRecipients(): loremipsum
getRecipients(): foo
getRecipients(): Bar
```

elttam

Primitives in Spring Framework

Classes:

InternetAddressEditor.java
MimeMessageHelper.java
MimeMailMessage.java
SimpleMailMessage.java

elttam

org.springframework.mail

- Root-level package for the Spring Framework’s email support.
  - **org.springframework.mail.javamail** → JavaMail support for Spring's mail infrastructure
    - InternetAddressEditor.java
    - MimeMessageHelper.java
    - MimeMailMessage.java
  - SimpleMailMessage.java

elttam

InternetAddressEditor.java

- Used for preparing an **InternetAddress** object using a supplied input email address.
- Simply “forwards” the input through to **InternetAddress(String)** constructor

InternetAddressEditor.java

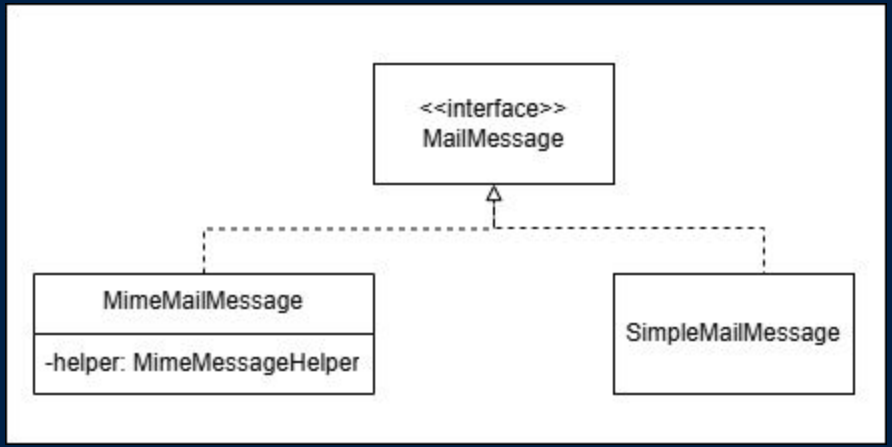
```java
@Override
public void setAsText(String text) throws IllegalArgumentException {
    if (StringUtils.hasText(text)) {
        try {
            setValue(new InternetAddress(text));
        }
        catch (AddressException ex) {
            throw new IllegalArgumentException("Could not parse mail address: " + ex.getMessage());
        }
    }
    else {
        setValue(null);
    }
}
```

InternetAddressEditor.java

```java
InternetAddressEditor editor = new InternetAddressEditor();
editor.setAsText("Alice <alice@example.com>");
InternetAddress address = (InternetAddress) editor.getValue(); // alice@example.com
```

```java
InternetAddressEditor editor = new InternetAddressEditor();
editor.setAsText("=?UTF-8?Q?Administrator_=3Cadmin@example.com=3E?= <attacker@evil.com>");
InternetAddress address = (InternetAddress) editor.getValue();
address.getPersonal(); // Administrator <admin@example.com>
address.getAddress(); // attacker@evil.com
```

MailMessage Interface



elttam

SimpleMailMessage.java

- Similar to **InternetAddressEditor**, setter methods accept encoded email strings that eventually make its way to **InternetAddress.parse()**
  - **setFrom()**
  - **setTo()**
  - ...

```
JavaMailSender emailSender = new JavaMailSenderImpl();
SimpleMailMessage message = new SimpleMailMessage();
message.setFrom("=?UTF-8?Q?Administrator_=3Cadmin@example.com=3E?= <attacker@evil.com>");
emailSender.send(message);
```

elttam

SimpleMailMessage.java

JavaMailSenderImpl.java

```
70   public class JavaMailSenderImpl implements JavaMailSender {
309      @Override
310      public void send(SimpleMailMessage... simpleMessages) throws MailException {
311          List<MimeMessage> mimeMessages = new ArrayList<>(simpleMessages.length);
312          for (SimpleMailMessage simpleMessage : simpleMessages) {
             MimeMailMessage message = new MimeMailMessage(createMimeMessage());
314              simpleMessage.copyTo(message);
315              mimeMessages.add(message.getMimeMessage());
316          }
```

Reader Mode

Threads & Variables | Server | Tomcat Catalina Log | Tomcat Localhost Log

"http-nio-90...ain": RUNNING

```
message.getMimeMessage().getFrom()[0].getPersonal()
```

send:315, JavaMailSenderImpl (org.spri

result = "Administrator <admin@example.com>"

elttam

MimeMailMessageHelper.java & MimeMailMessage.java

- **MimeMailMessage** represents the email envelope object
- **MimeMailMessageHelper** is used to populate the various fields of the MimeMessage
  - **MimeMailMessageHelper(MimeMessage, String))**
    - Does the application pass in a user-controlled String?

elttam

MimeMailMessageHelper.java & MimeMailMessage.java

```java
Session session = Session.getInstance(new Properties());
MimeMessage mimeMessage = new MimeMessage(session);
MimeMessageHelper helper = new MimeMessageHelper(mimeMessage, "utf-16");

helper.setFrom("alicé <alice@example.com>");
```

```
toString(): =?utf-16?Q?=FE=FF=00a=00l=00i=00c=00=E9?= <alice@example.com>
getAddress(): alice@example.com
getPersonal(): alicé
```

Hibernate Validator

A quick look at the **@Email** annotation

elttam

@Email Annotation

- Validates that the string is an email address
- Not RFC 2047 compliant (no encoded strings!)
- Optional custom regex via **@Email(regexp="INPUT")**

```java
class User {

    @Email
    private String email;

    public User(String email) {
        this.email = email;
    }

    public String getEmail() {
        return email;
    }
}
```

elttam

@Email Annotation

- Default regex gets pretty intense!
- Validates the local part (before “@”) and domain part (after “@”)

```
private static final String LOCAL_PART_ATOM = "[a-z0-9!#$%&'*+/=?^_`{|}~\u0080-\uFFFF-]";
private static final String LOCAL_PART_INSIDE_QUOTES_ATOM = "(?:[a-z0-9!#$%&'*.(),<>\\[\\]:;  @+/=?^_`{|}~\u0080-
\uFFFF-]|\\\\\\\\\|\\\\\\\\")";
/**
 * Regular expression for the local part of an email address (everything before '@')
 */
private static final Pattern LOCAL_PART_PATTERN = Pattern.compile(
        "(?:" + LOCAL_PART_ATOM + "+|\"" + LOCAL_PART_INSIDE_QUOTES_ATOM + "+\")" +
                "(?:\\." + "(?:" + LOCAL_PART_ATOM + "+|\"" + LOCAL_PART_INSIDE_QUOTES_ATOM + "+\")" + ")*",
        CASE_INSENSITIVE
```

elttam

@Email Annotation

- **"foo@bar.com@"@example.com** will pass the default regex validation checks!
  - Exact implication depends on how the application uses the input email
  - e.g: **.split("@")[1]** → **bar.com**

elttam

Conclusion

elttam

Any questions?

Thank you