

# Java MAIL



# 1.INTRODUCTION

**Format d'un message électronique**

# 1.INTRODUCTION: Format d'un message électronique

## A) Format de base: (MFS =Message Format Standard - RFC 822)

Texte simple et pas de pièce attachée

<code>&lt;message&gt; =</code>	<code>&lt;en-tête du message [<i>message header fields</i>] &gt;</code> <code>CR LF</code> <code>&lt;corps du message [<i>message body</i>] &gt;</code>
--------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------

### • Série de différents headers de MFS:

Clés : <valeurs> = hashtable

1) Origine: **FROM:** <expéditeur>  
**DATE:** <date de soumission>

2) Destination: **TO:** <destinataire(s)>  
**Cc:** <destinataire(s)>  
**Bcc:** <destinataire(s)>

champs header  
ou header fields

<nom du header> : <valeur du header> CR-LF  
Série de headers qui a été repris par le protocole  
HTTP plus tard

# 1.INTRODUCTION: Format d'un message électronique

Identification unique de chaque message: le plus souvent, il comporte plutôt une partie unique sur Internet, comme l'identification de la machine, et une partie unique sur cette machine

3) Identification: **Message-Id :** <identificateur unique (sur INTERNET)>

4) Information: **Subject:** <texte sujet du message>  
**Content-type:** <code MIME>

*MIME: Multipurpose Internet Mail Extensions*

**Text/plain; charset=US-ASCII**  
Pour MFS. Utilisé aussi en HTTP

# 1.INTRODUCTION: Format d'un message électronique

Type	Définition	Quelques sous-titres courants
text	information textuelle	<ul style="list-style-type: none"><li>♦ plain : texte non formaté – aucun software complémentaire n'est nécessaire pour appréhender l'entièreté du message – il est possible de spécifier le jeu de caractères par une clause charset : Content-type: text/plain; charset=us-ascii Content-type: text/plain; charset=iso-885-5</li><li>♦ html : le texte comporte des tags HTML</li></ul>
Image	formats gif ou jp(e)g	<ul style="list-style-type: none"><li>♦ gif</li><li>♦ jpeg</li></ul>
Vidéo		<ul style="list-style-type: none"><li>♦ mpeg et autres</li></ul>
application	informations binaires, à priori destinées à être exécutées par une application basée sur la messagerie	<ul style="list-style-type: none"><li>♦ <b>octet-stream</b> : données binaires brutes – c'est le sous-type utilisé lorsqu'une application client ne connaît pas le type/sous-type précisé dans le message reçu</li><li>♦ postscript : pour transporter des informations codées en postscript</li></ul>

# 1.INTRODUCTION: Format d'un message électronique

Type	Définition	Quelques sous-titres courants
message	message faisant partie d'un autre message et susceptible de contenir des informations d'une autre nature que celle du message principal	<ul style="list-style-type: none"><li>♦ rfc822 : selon la RFC de base de la messagerie électronique</li></ul> <p>On est maintenant dans les RFC de l'ordre de 5xxx =&gt; 822 est très vieux</p>
multipart	principal ♦ rfc822 : selon la RFC de base de la messagerie électronique multipart données composées de groupes d'informations de types différents	<ul style="list-style-type: none"><li>♦ <u>mixed</u> : les différentes composantes sont indépendantes et doivent être fournis dans un ordre précis</li><li>♦ alternative : les différentes composantes sont des versions différentes de la même information – le choix se fera selon les préférences déclarées du récepteur</li><li>♦ parallel : l'ordre des différentes parties n'a pas d'importance</li><li>♦ related : le message est une page WEB consistant en son code HTML, mais aussi en les ressources qu'il référence.</li></ul>

Outlook envoie tjs un multipart (jamais du 822) avec 2 morceaux en /plain et /html

# 1.INTRODUCTION: Format d'un message électronique

5) Headers propriétaires: **X-<nom\_propriétaire>: <valeur au sens propriétaire>**

Le champ Content-Type peut encore utiliser des types propriétaires, référencés par "X-... (extended)", encore qu'il soit plutôt recommandé de définir des sous-types des 7 types existants (référéncés par "x-...")

<b>MFS</b>	
<b>Nom complet :</b>	<b>Internet Message Format Standard</b>
<b>Nature :</b>	<b>Protocole de spécification de la nature des données transmises</b>
<b>RFC :</b>	<b>822-2076</b>
<b>Utilisé par :</b>	<b>SMTP, POP3</b>
<b>Champs courants</b>	<b>Date, From, To, Cc, Subject</b>

# 1.INTRODUCTION: Format d'un message électronique

## **B) Les spécifications MIME** ( Multipurpose Internet Mail Extensions)

Très utile dans le cas de messages évolués

<b>MIME</b>	
Nom complet :	Multipurpose Internet Mail Extensions
Nature :	Protocole de spécification de la nature des données transmises
RFC :	2045-2046-2047-2048-2049
Utilisé par :	HTTP, SMTP, POP3
Types courants :	text/plain, text/html, application/octet-stream
Codages courants :	7bit, quoted-printable, base64



# 1.INTRODUCTION: Format d'un message électronique

1) le champ de version :

**MIME-Version** : <version de MIME>

2) le champ du type du contenu du message :

**Content-Type** : <type>/<sous-type> [; informations complémentaires>]

Spécifie si le contenu est altérable ou pas en passant d'un serveur à l'autre lors du transfert du message.

3) Les encodages (optionnel)

**Content-Transfer-Encoding**: <type d'encodage>

- 7 bits, 8 bits et binary
- Quoted-printable
- **Base64** (par exemple: les certificats (PEM) sous forme de fichiers particuliers (encoding particulier) => ne peut être modifier)

# 1.INTRODUCTION: Format d'un message électronique

## 7bit, 8bit et Binary :

il s'agit des représentations en ASCII 7 bits (américain), ASCII 8 bits et chaîne d'octets. En fait, il n'y donc aucun codage particulier (on parle encore d'identity) et les logiciels de transfert (les MTAs) peuvent donc réaliser leur propre encodage selon les besoins de transport sur le réseau.

## Quoted-printable : (la plupart des messages, mais attention au charset)

il s'agit de données sous forme de texte classique, si ce n'est que:

- les octets sont codés en **big-endian**;
- les caractères autres que ceux de code ASCII 33 à 60 et 62 à 126 ainsi que toutes les formes d'espaces (c'est-à-dire 9 et 32) doivent être **codés sous forme d'un signe "=" suivi de leur valeur hexadécimale (sur deux digits)**; bien clairement, ce type d'encodage concerne les messages textes utilisant des caractères non anglophones. Comme le résultat du codage est spécifique (ce n'est plus du texte de base), le message, bien que plus ou moins lisible, ne sera **pas modifié par le processus de transport MFS** standard.

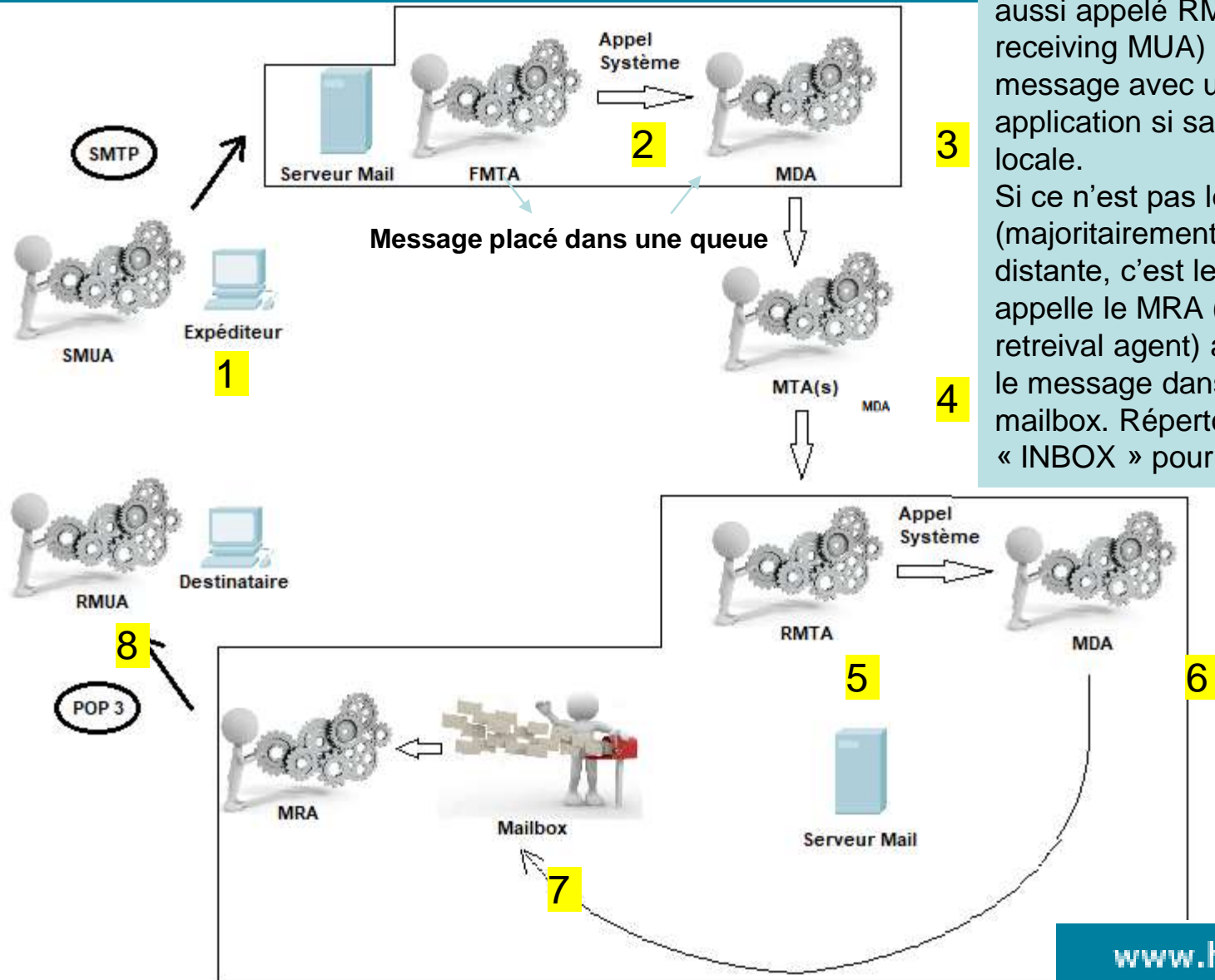
# 1.INTRODUCTION: Format d'un message électronique

## Base64 :

Décompose les groupes successifs de 24 bits (3 octets supposés être en bigendian) en 4 groupes de 6 bits. Chacun de ces groupes est alors envoyé en tant que caractère ASCII, selon la convention que "A" correspond à 0, "B" à 1, ..., "Z" à 25, "a" à 26, ..., "z" à 51, "0" à 52, "9" à 61, "+" à 62 et "/" à 63. On utilise donc un jeu de 64 caractères. Comme la fin du message peut contenir des octets vides, ceux-ci sont remplis (padding) avec "==" (il n'y a plus que deux groupes de 6 bits) et "=" (il n'y a plus que trois groupes de 6 bits). Il s'agit évidemment du format destiné à tout ce qui n'est pas texte devant être compris tel quel par le destinataire (comme une image, un programme compilé, etc).

A nouveau, le résultat du codage est spécifique (ce n'est plus du texte de base) et le message, cette fois totalement illisible, ne sera pas modifié par le processus de transport MFS standard.

## 1.2 INTRODUCTION: Le modèle classique d'une messagerie



Le MUA (mail user agent) aussi appelé RMUA (receiving MUA) lit alors le message avec une application si sa mailbox est locale. Si ce n'est pas le cas (majoritairement) => mailbox distante, c'est le MUA qui appelle le MRA (mail retrieval agent) afin de lire le message dans la mailbox. Répertoire appelé « INBOX » pour POP3

## P1.3 INTRODUCTION. Quelques éléments du protocole SMTP

SMTP	
Nom complet :	Simple Mail Transfer Protocol
Nature :	Protocole applicatif de transfert de messages électroniques
RFC :	821-2821
Port par défaut :	25
Protocole de transport :	TCP
Principales commandes :	helo, mail from, rcpt to, data, quit

**Basé sur TCP, assure la fiabilité**

## P1.3 INTRODUCTION. Quelques éléments du protocole SMTP

Commandes (toujours 4 lettres + param)	Description
<b>helo</b> <adresse IP machine expéditeur> <b>helo</b> <nom machine expéditeur>	Permet de s'identifier en tant que machine émettrice
<b>mail from:</b> <adresse e-mail de l'expéditeur>	Permet de spécifier l'adresse e-mail de l'émetteur
<b>rcpt to:</b> <adresse e-mail du destinataire>	Permet de spécifier l'adresse e-mail du récepteur
<b>data</b>	Représente le corps du message – il se termine par un point isolé sur une ligne
<b>quit</b>	Envoie tous les messages mis en attente dans la file d'envoi
<b>rset</b>	Permet d'annuler le message en cours de transaction.
<b>noop</b>	Opération vide = "j'existe" ... . C'est une opération vide (est également un point de retour en assembleur. Pas de tps à perdre => déconnexion rapide (timeout très court) => pas de connexion permanente sauf en envoyant des <b>noop(s)</b> .
<b>help</b>	Pour obtenir la liste des commandes SMTP supportées par le serveur

## P1.3 INTRODUCTION. Quelques éléments du protocole SMTP

Code	Signification
214	Message d'aide
220	Service disponible
221	Canal de transmission en cours de fermeture
250	Commande reçue exécutée avec succès
251	Utilisateur non local avec relais automatique – le message est réémis vers l'adresse
354	Début de l'encodage d'un message
450	Erreur : boîte aux lettres non disponible (l'accès est impossible ou bloqué)
451	Erreur : erreur dans le traitement local du message
452	Erreur : pas assez de ressources système
500	Erreur : faute de syntaxe dans la commande
501	Erreur : faute de syntaxe dans les paramètres de la commande
502	Erreur : commande non implémentée
504	Erreur : paramètre de commande non implémenté
550	Erreur : boîte aux lettres inaccessible
551	Erreur : utilisateur non local sans relais automatique
552	Erreur : manque de ressource de stockage
553	Erreur : nom de boîte aux lettres non autorisé

Ressemble aux codes de HTTP: ce dernier a copié SMTP


## P1.3 INTRODUCTION. Quelques éléments du protocole SMTP

### Exemple de transaction SMTP en ligne de commandes

**Telnet:** port par défaut : 23 Protocole de transport : TCP

Connexion au serveur mail de tests (non secure !) de l'école: u2 (10.59.26.134)

**telnet u2 25** ou **telnet 10.59.26.134 25** (dans notre cas utilisation de Putty, un invite de commande suffit déjà)

 u2.tech.hepl.local - PuTTY

```
220 u2.tech.hepl.local ESMTP Postfix (Ubuntu)
HELO u2
502 5.5.2 Error: command not recognized
HELO u2
250 u2.tech.hepl.local
MAIL FROM:paeme@u2.tech.hepl.local
250 2.1.0 Ok
RCPT TO:paeme@u2.tech.hepl.local
250 2.1.5 Ok
DATA
354 End data with a dot (CRLF) on a line by itself
Hello mail
.
250 2.0.0 Ok: queued as D5281429E9
```

A la fin, le mail est placé par le MTA dans la queue (mail queue delivery)

Il faudra bien entendu, aller rechercher ce mail avec d'autres commandes (voir plus tard)



## P1.3 INTRODUCTION. Éléments principaux de la description d'un protocole

	SMTP	POP3	IMAP	HTTP	FTP	SNMP
<b>Spécification</b>	Couche transport (TCP, UDP) port(s) par défaut RFC(s) principale(s)			TCP 80 RFC: 2616 - 7231		
<b>Commandes</b>	Commandes de base / secondaire			GET/POST/HEAD PUT, ...		
<b>Trames</b>	Une ou plusieurs requêtes/réponses			Requête: start line + headers + SP + query-string		
<b>Etats</b>	Sans (SMTP) <b>ou</b> machine à états (POP3)			Sans		
<b>Erreurs</b>	Sigles ? Codes d'erreurs avec classifications? (SMTP à des codes d'erreurs mais pas POP3 soit ok ou nok)			codes en 5 groupes: 1: ... 2: ... 3: ... 4: ... 5: ...		

## **2. LES E-MAILS ET LES PACKAGES JAVAX**

## 2.1 les packages

- JavaMail est composé d'un ensemble d'APIs (classes + interfaces)
- Il fait partie de J2EE.
- Répartition en 4 packages: (le tout dans **mail.jar**)
  - **Javax.mail**: Les classes de base qui permettent de réaliser les opérations de base des mails.
  - **Javax.mail.event**: Événements associés aux mails (du type arrivée de courrier) et listeners correspondants.
  - **Javax.mail.internet**: Les classes spécifiques aux mails Internet.
  - **Javax.mail.search**: Les fonctionnalités de recherche dans les mails.
- Ensemble de composants génériques (programmation indépendante de la plate-forme) => interfaces + classes abstraites.
- Des providers doivent fournir les implémentations correspondantes: `javamail.providers` et `javamail.default.providers`

## 2.1 les packages

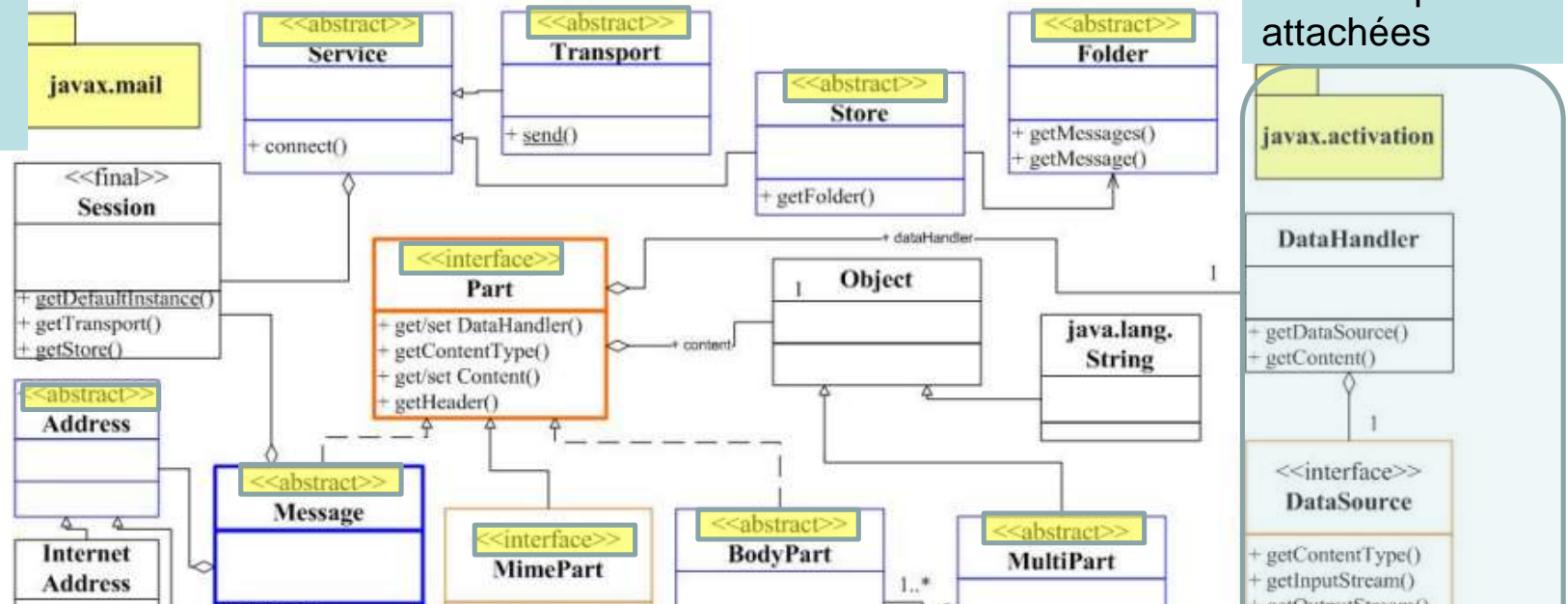
- Ces implémentations sont censées comporter:
  - un protocole de transfert [transport] de courrier électronique - habituellement, il s'agit de SMTP,
  - un protocole de stockage et de remise du courrier [store] - habituellement, il s'agit de POP3 ou IMAP.
- Le fichier mail.jar contient un fichier javamail.default.providers

### javamail.default.providers

- **JavaMail IMAP** provider Sun Microsystems, Inc protocol=imap; type=store; class=com.sun.mail.imap.**IMAPStore**; vendor=Sun Microsystems, Inc;
- **JavaMail SMTP** provider Sun Microsystems, Inc protocol=smtp; type=transport; class=com.sun.mail.smtp.**SMTPTransport**; vendor=Sun Microsystems, Inc;
- **JavaMail POP3** provider Sun Microsystems, Inc protocol=pop3; type=store; class=com.sun.mail.pop3.**POP3Store**; vendor=Sun Microsystems, Inc;

## 2.2 Le package javax.mail (diagramme des classes)

De base  
seulement  
javax.mail



Pour les pièces  
attachées

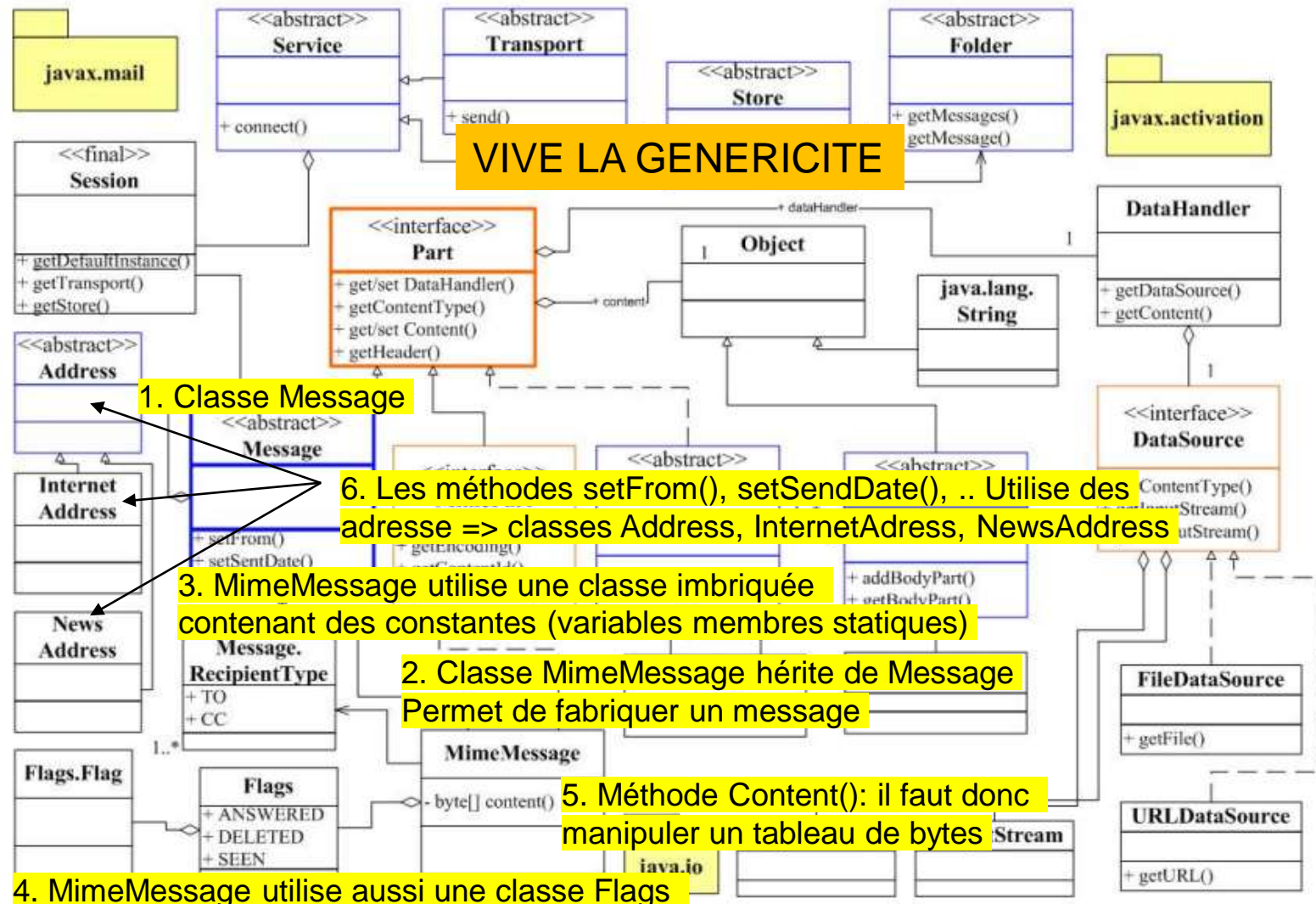
Un provider fournit donc les implémentations de ces interfaces et classes abstraites:

- D'un Part
- D'un Message
- .....

javamail.default.providers (implémentation par défaut)



## 2.2 Le package javax.mail (diagramme des classes) Définition de l'environnement



1. Classe Message

6. Les méthodes `setFrom()`, `setSendDate()`, .. Utilise des adresse => classes `Address`, `InternetAddress`, `NewsAddress`

3. `MimeMessage` utilise une classe imbriquée contenant des constantes (variables membres statiques)

2. Classe `MimeMessage` hérite de `Message`  
Permet de fabriquer un message

5. Méthode `Content()`: il faut donc manipuler un tableau de bytes

4. `MimeMessage` utilise aussi une classe `Flags`

(classe `Flags.flag` - flag est une classe imbriquée) contenant des flags ;-) message lu, effacé, ..

## 2.2 Le package javax.mail (diagramme des classes)

### La classe message / interface Part ...

Au départ, on a un message ne contenant que du texte (= tableau de bytes Byte content() ) mais ensuite un message pourrait être constitué de plusieurs parties (à prévoir)=> l'interface PART. Une partie = Part (exemple : des pièces attachées).

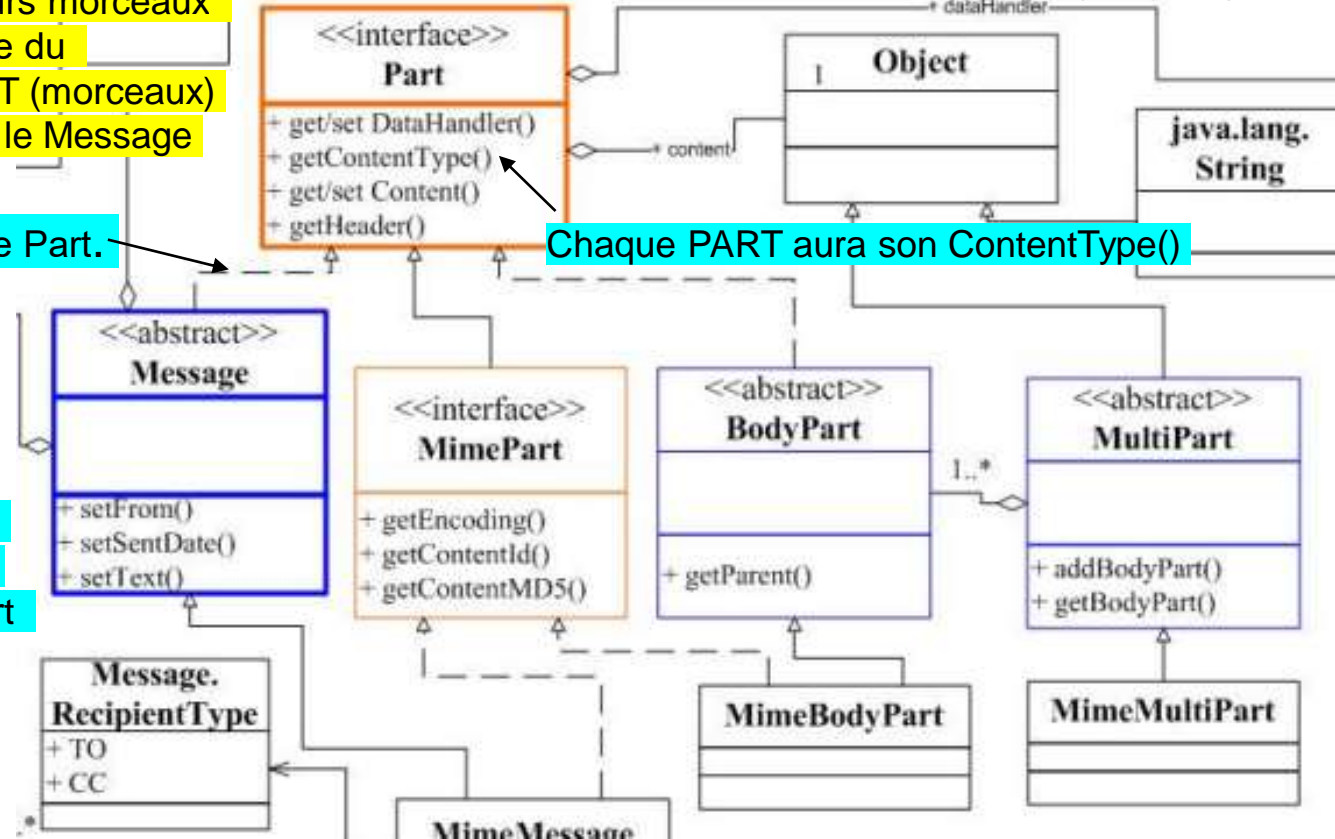
Si un message contient plusieurs morceaux pas la peine de placer l'adresse du Destinataire dans chaque PART (morceaux) Mais seulement, une fois dans le Message

Chaque PART contiendra un contenu d'un certain type: image, texte..

Message implémente l'interface Part.

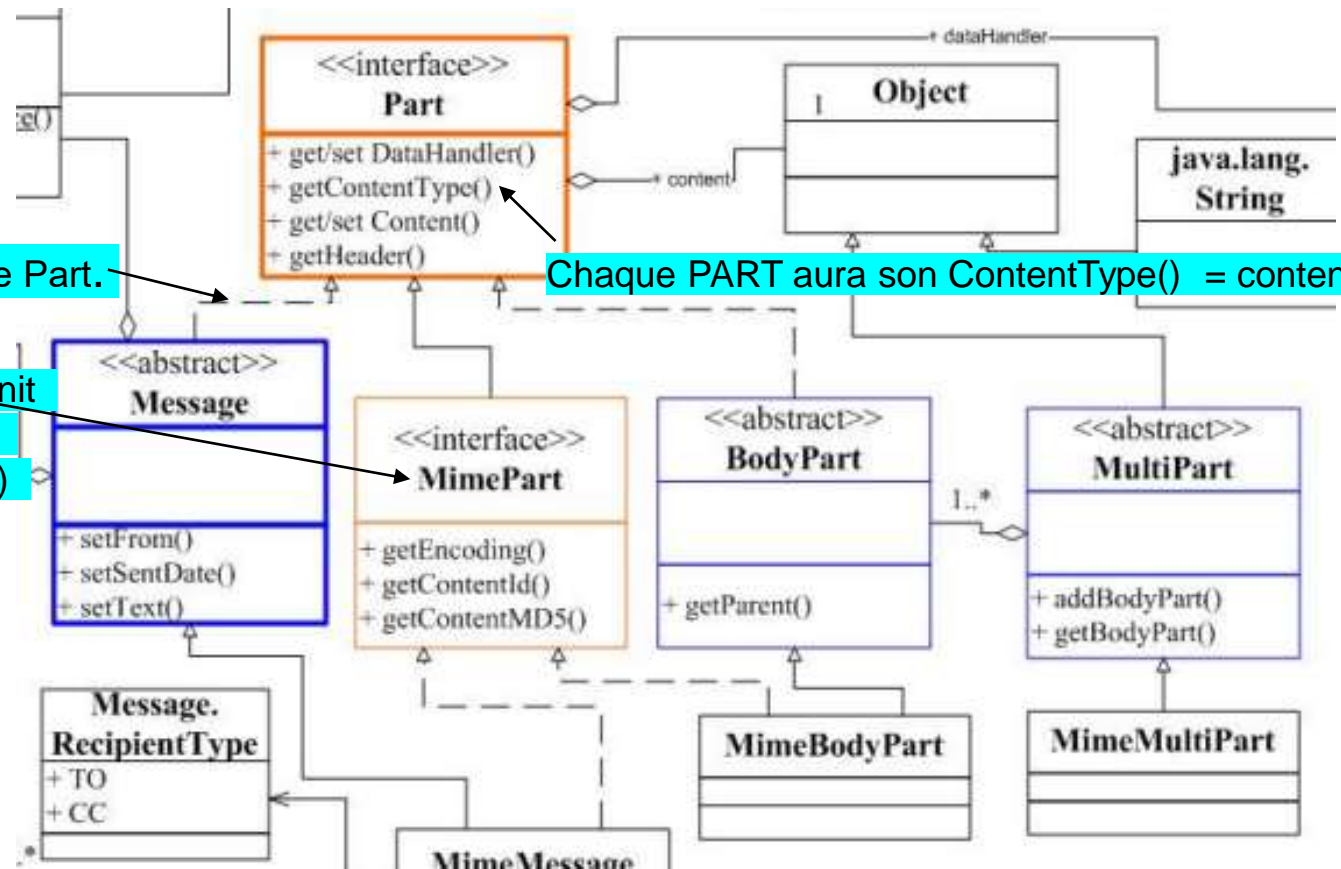
Chaque PART aura son ContentType()

Un Message de type RFC 822 (simple texte), contient un seul Part => un message est un Part avec son propre contenu et headers définis par l'interface Part.



## 2.2 Le package javax.mail (diagramme des classes)

### La classe message / interface Part ...



Message implémente l'interface Part.

Chaque PART aura son ContentType() = contenu

MimePart dérive de Part et définit l'encoding et le **Content MD5()** (= digest, contrôler l'intégrité)



## 2.2 Le package javax.mail (diagramme des classes)

### La classe transport, Folder, ...

Il faut pouvoir définir le MTA/MDA => il faut les définir dans des propriétés et utiliser des fonctions pour les envois et la lecture => classe abstraite Transport (mauvais nom, aurait dû s'appeler Transfert) contient la méthode send()

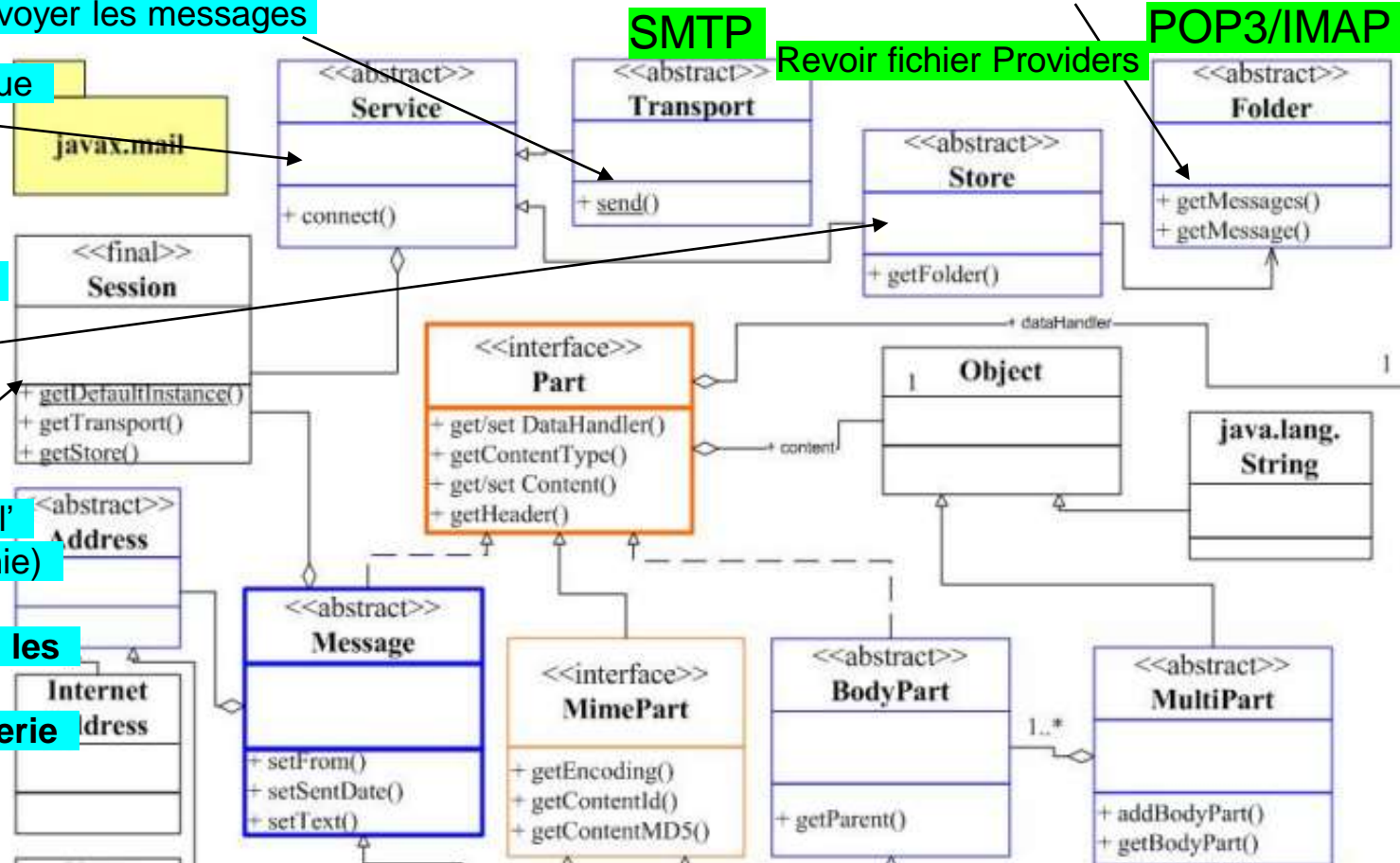
Méthodes getMessage(s)() pour aller rechercher un/les message(s)

Méthode send() pour envoyer les messages

La classe Service effectue la connexion (encapsule: socket)

Avant de récupérer un message, il faut d'abord ouvrir la mailbox: (classe Store + méthode getFolder())

Une classe session 'final' (ne peut pas être redéfinie) permet de lier le tout. Cet objet contient tous les renseignements du contexte de la messagerie.

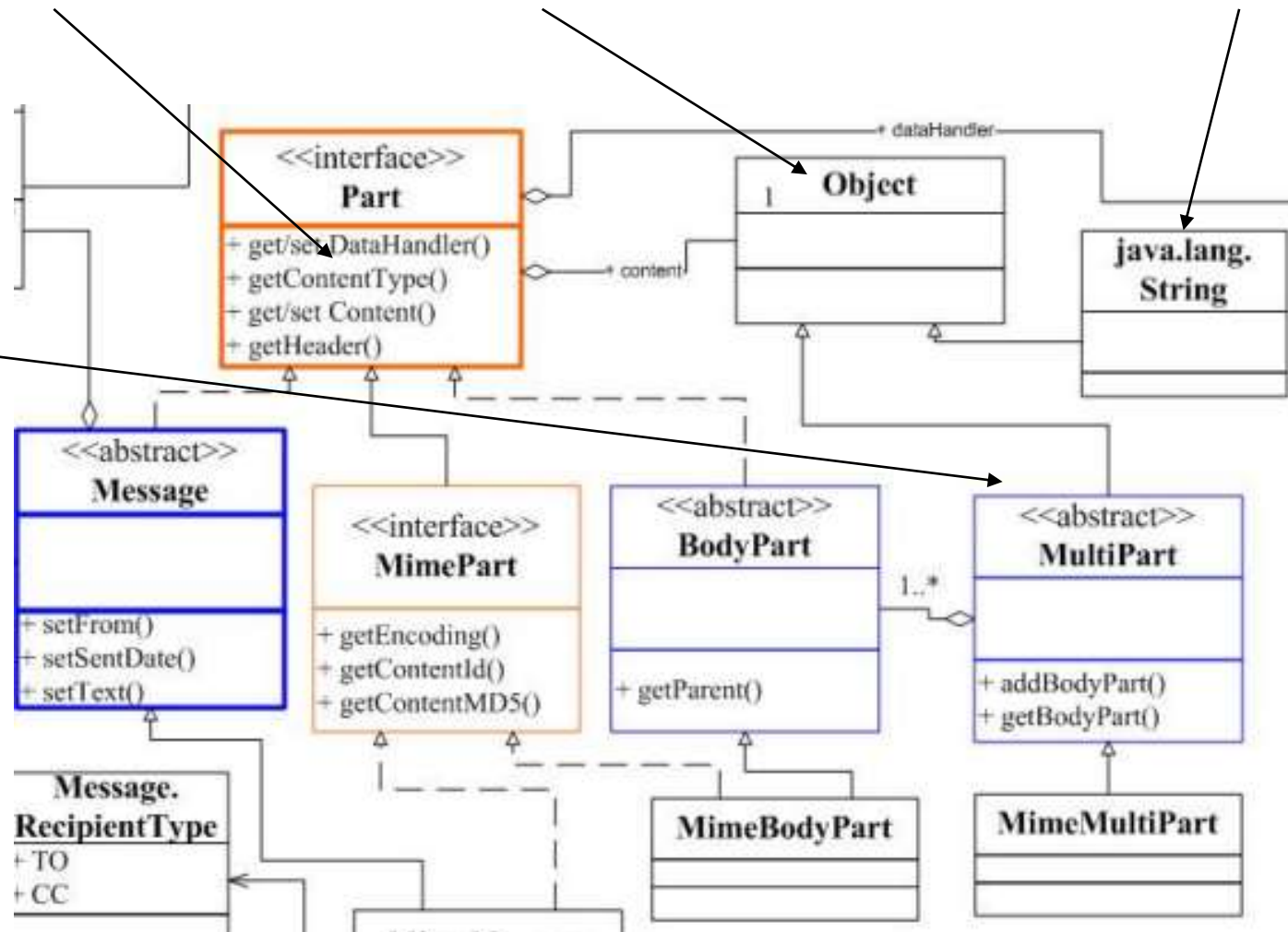


## 2.2 Le package javax.mail (diagramme des classes)

### Le contenu des messages pardon des Parts

Part contient une méthode getContent() => le contenu est un Object et dans le cas le plus simple est de type String

Le contenu peut être aussi un Multipart = ensemble de PART (BodyPart ?)



### 3. CONSTRUCTION D'UN MESSAGE SIMPLE

### 3.1 Programmation de l'envoi d'un simple message texte

classe Session :

```
public static Session getInstance(java.util.Properties props)
```

classe **System** :

```
public static Properties getProperties()
```

## Properties = Format d'une Hashtable

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

```
import javax.mail.internet.*:
```

**import** Création d'une **classe** à la « **volée** ». Exemple pour Gmail

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

Session session = Session.getInstance(props,
    new javax.mail.Authenticator() {
```

public class **@Override**

```
protected PasswordAuthentication getPasswordAuthentication() {  
    return new PasswordAuthentication(username, password);  
}
```

```
public void ... }
public void ... }));
```

**Propriétés du système**

```
Properties prop = System.getProperties();
```

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

```
Session sess = Session.getDefaultInstance(prop, null);
```

~~password (si secure -> autres ports.~~

exemple: **smtp-relay.gmail.com**: 25, 465 ou 587.)

## Méthode statique afin d'obtenir l'objet session

Authentificateur pas nécessaire pour u2 par contre gmail en a besoin

www.hep.be

# 3.1 Programmation de l'envoi d'un simple message texte

## 1) L'interface Part.

### a) Les attributs:

Dont notamment:

```
public java.lang.String getContentType() throws MessagingException
```

Le type est renvoyé selon la codification MIME (on obtient null si le type ne peut être déterminé). D'ailleurs, la méthode

```
public boolean isMimeType(java.lang.String mimeType) throws  
MessagingException
```

permet de vérifier que l'on a affaire à un type standard, donc pas un type propriétaire.

# 3.1 Programmation de l'envoi d'un simple message texte

b) Un contenu:

- un **DataHandler** (classe définie dans javax.activation), obtenu par la méthode

```
public javax.activation.DataHandler getDataHandler()  
    throws MessagingException
```

- un **flux d'entrée**, obtenu au moyen de la méthode prévisible

```
public java.io.InputStream getInputStream()  
    throws java.io.IOException, MessagingException
```

- un **objet Java**, instance d'une classe quelconque et qui pourra fournir son contenu

```
public java.lang.Object getContent()  
    throws java.io.IOException, MessagingException.
```

# 3.1 Programmation de l'envoi d'un simple message texte

## 2) Les classes messages.

### a) La classe de base

La classe abstraite attendue **Message** dérivée directement d'Object et implémente l'interface Part.

```
public void setText(java.lang.String text) throws MessagingException
```

Les méthodes:

- `public abstract void setFrom (Address address)  
throws MessagingException`
- `public void setRecipient(Message.RecipientType type,  
Address address) throws MessagingException`
- `public abstract void setSubject(java.lang.String subject)  
throws MessagingException`
- `public abstract void setSentDate(java.util.Date date)  
throws MessagingException`

# 3.1 Programmation de l'envoi d'un simple message texte

Deux classes supplémentaires:

- Address: **InternetAddress** et **NewsAddress**  
(du package javax.mail.internet)
- RecipientType: **public static final Message.RecipientType TO**  
**public static final Message.RecipientType CC**  
**public static final Message.RecipientType BCC**

L'objet message:

(non composite – on dit encore "**Simple part**", par opposition à "Multipart") sous cette forme :

Header	
	Attributs définis dans Part, dont Content-Type
	Attributs ajoutés par Message, dont From, To, Subject
Corps [ <i>Content body</i> ]	
	Objet DataHandler donnant accès au contenu du message



# 3.1 Programmation de l'envoi d'un simple message texte

La classe classique:

. Une classe dérivée de **Message**: **MimeMessage** (définie dans le package `javax.mail.internet`) implémente effectivement les messages classiques utilisant la codification MIME. Elle implémente d'ailleurs l'interface **MimePart**.

Header	
	Attributs définis dans Part, dont Content-Type
	Attributs ajoutés par Message, dont From, To, Subject
	Attributs ajoutés par MimeMessage, dont les flags
Corps [ <i>Content body</i> ]	
	Objet DataHandler donnant accès au contenu du message

## 3.1 Programmation de l'envoi d'un simple message texte

Envoi d'un message.

La classe **Transport** est une classe dérivée de la classe **Service**

Elle contient la méthode:

```
public static void send(Message msg) throws MessagingException
```

## 3.2 Programmation de l'envoi d'un simple message texte: (SMTP)

```
try
{
    String exp = "vilvens@u2.tech.hepl.local";
    String dest = "charlet@u2.tech.hepl.local";
    String sujet = "Premier essai";
    String texte = "Essai avec JavaMail - Peux-tu me répondre ? - Merci - CV";
    MimeMessage msg = new MimeMessage (sess);
    msg.setFrom (new InternetAddress (exp));
    msg.setRecipient (Message.RecipientType.TO, new InternetAddress (dest));
    msg.setSubject(sujet);
    msg.setText (texte);
    System.out.println("Envoi du message");
    Transport.send(msg);
    System.out.println("Message envoyé");
}
catch (MessagingException e)
{
    System.out.println("Erreur sur message : " + e.getMessage());
}
catch (Exception e)
{
    System.out.println("Erreur sur message : " + e.getMessage());
}
```

Headers

Comme l'objet **msg** a été instancié à partir de classe **MimeMessage** avec comme paramètre l'objet **session** représente le contexte, il contient dans ses propriétés les informations concernant le MTA => pas besoin de le rappeler lorsqu'on utilise la **méthode statique: send()** de la classe **Transport**

# 3.3 Programmation de la réception d'un simple message texte: POP3

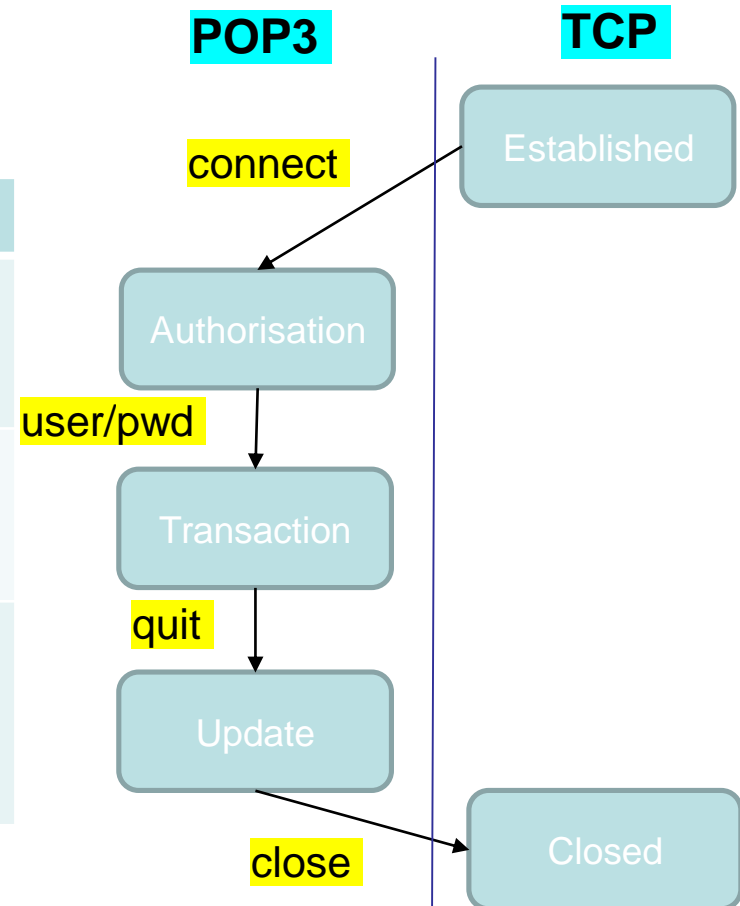
## Le protocole POP3 (Post Office Protocol version 3)

POP3	
Nom complet :	Post Office Protocol version 3
Nature :	Protocole applicatif de récupération des messages électroniques
RFC :	1725
Port par défaut :	110
Protocole de transport :	TCP
Principales commandes :	user, pass, retr, dele

# 3.3 Programmation de la réception d'un simple message texte: POP3

**POP3 un protocole à états (3)** contrairement à SMTP: On est notamment obligé de passer par un état « AUTHORISATION » (authentification).

Etats	Signification
AUTHORIZATION	La connexion TCP étant acceptée, le client doit à présent s'authentifier
TRANSACTION	La boîte aux lettres de l'utilisateur authentifié est verrouillée. Le client peut consulter, supprimer, etc.
UPDATE	Toutes les modifications à la boîte aux lettres sont enregistrées; celle-ci est ensuite déverrouillée et la connexion TCP est fermée.

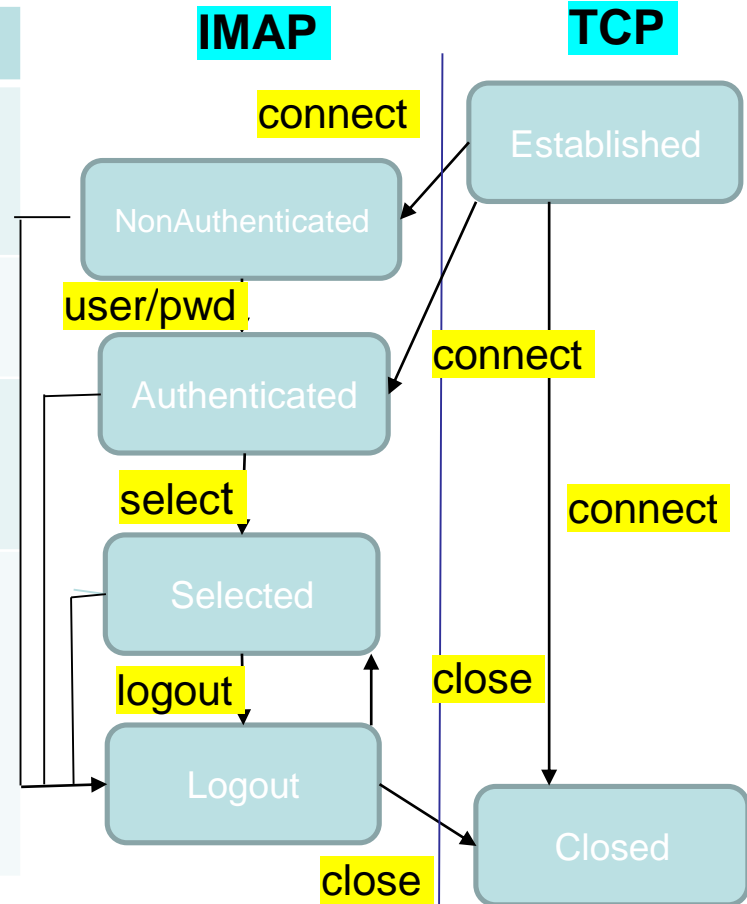


# 3.3 Programmation de la réception d'un simple message texte: IMAP

**Comparaison POP3 et IMAP point de vue « états »**

**IMAP un protocole à états (4)** car contrairement à POP3, il peut y avoir plusieurs mailbox => sélectionner la mailbox

Etats	Signification
NON_AUTHENTICATED	La connexion TCP étant acceptée, le client doit à présent s'authentifier
AUTHENTICATED	Le client s'est connecté et fait reconnaître par le serveur.
SELECTED	Le client a sélectionné la boîte aux lettres sur laquelle il va effectuer diverses opérations.
LOGOUT	Dès que la connexion au serveur est perdue, soit parce que le client a terminé, soit parce que le serveur refuse ses services, ou encore parce que la connexion a été coupée.



# 3.3 Programmation de la réception d'un simple message texte: POP3

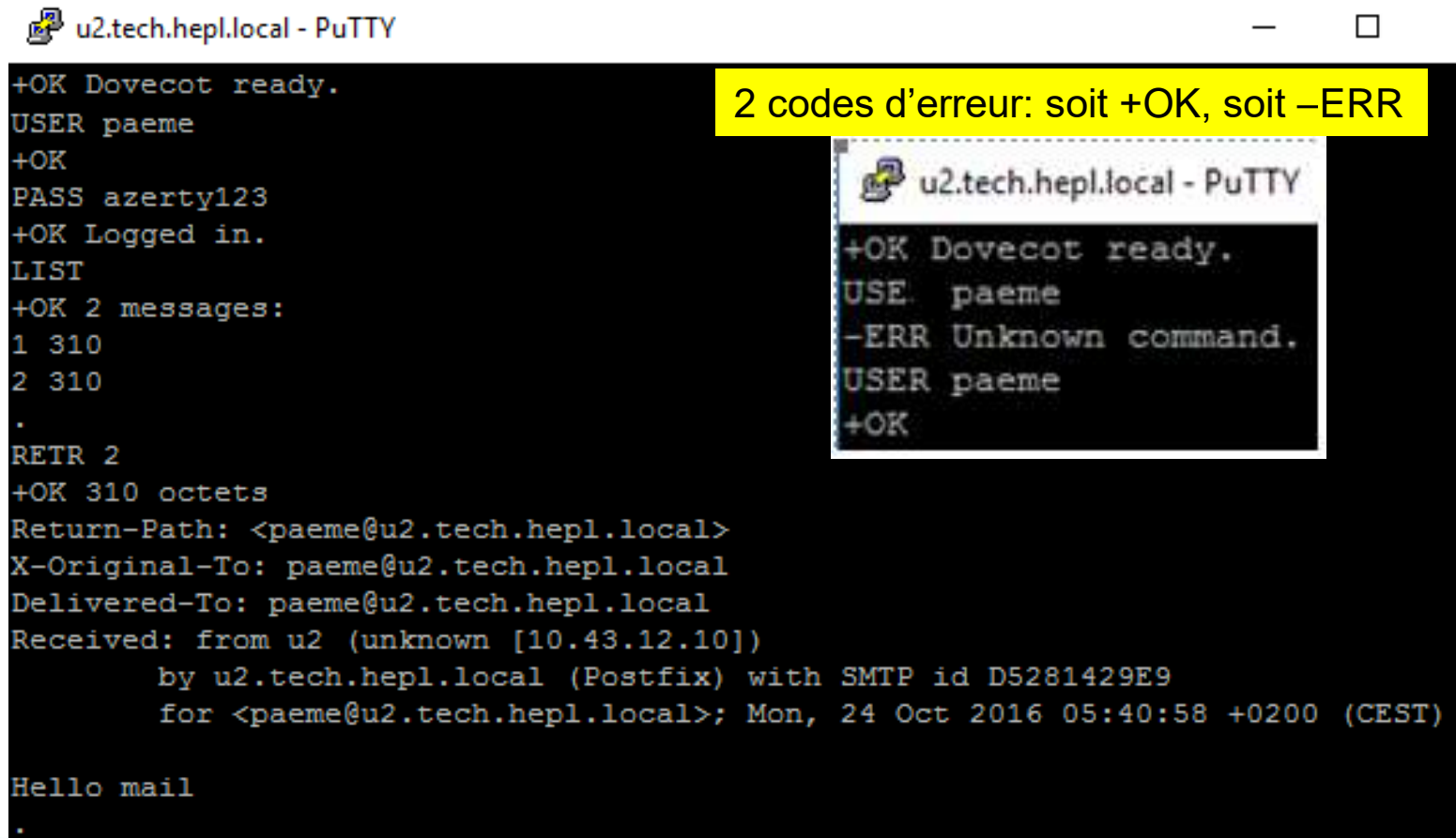
Les commandes de base (CLI), toujours composées de 4 lettres

Commandes	Description
<b>user</b> <nom utilisateur>	Permet de s'identifier comme utilisateur
<b>pass</b> <pwd>	Permet de confirmer son identité par introduction du mot de passe
<b>list</b>	Liste les messages (avec numéro et taille)
<b>retr</b> <numéro du message>	Permet de récupérer le message du numéro précisé
<b>dele</b> <numéro du message>	Permet de supprimer le message du numéro précisé (plus exactement, le message est marqué logiquement pour la suppression)
<b>top</b> <numéro du message><nbr de ligne>	Permet d'obtenir les n premières lignes du message
<b>stat</b>	Donne le nombre et le taille cumulée des messages non lus
<b>quit</b>	On passe dans l'état UPDATE et les messages marqués pour la suppression le sont alors effectivement.
<b>rset</b>	Permet d'annuler la suppression de tous les messages

# 3.3 Programmation de la réception d'un simple message texte: POP3

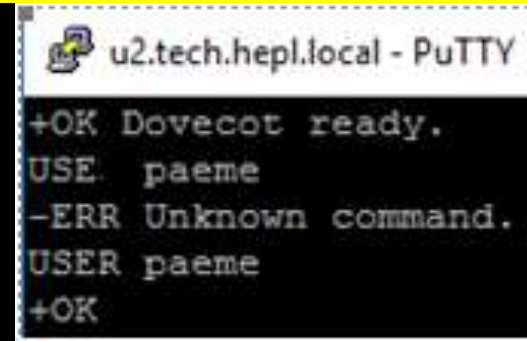
## Exemple de transaction POP3 en ligne de commande

telnet u2 110 ou telnet 10.59.26.134 110 (serveur mail de tests école)



```
u2.tech.hepl.local - PuTTY
+OK Dovecot ready.
USER paeme
+OK
PASS azerty123
+OK Logged in.
LIST
+OK 2 messages:
1 310
2 310
.
RETR 2
+OK 310 octets
Return-Path: <paeme@u2.tech.hepl.local>
X-Original-To: paeme@u2.tech.hepl.local
Delivered-To: paeme@u2.tech.hepl.local
Received: from u2 (unknown [10.43.12.10])
    by u2.tech.hepl.local (Postfix) with SMTP id D5281429E9
    for <paeme@u2.tech.hepl.local>; Mon, 24 Oct 2016 05:40:58 +0200 (CEST)
Hello mail
.
```

2 codes d'erreur: soit +OK, soit -ERR



```
u2.tech.hepl.local - PuTTY
+OK Dovecot ready.
USE paeme
-ERR Unknown command.
USER paeme
+OK
```



## 3.3 Programmation de la réception d'un simple message texte: POP3

Tous les message arrivent dans **un folder**. La classe suivante permet de gérer ce folder.

- Méthode de classe Store (dérivé de Service)

```
public Store getStore(java.lang.String protocol)  
                    throws NoSuchProviderException
```

- La relation avec le serveur POP3 est effectué à l'aide de la méthode héritée de Service suivante:

```
public void connect(java.lang.String host, java.lang.String user, java.lang.String  
password thows MessagingException
```

## 3.3 Programmation de la réception d'un simple message texte: POP3

- Créer un objet instanciant la classe **Folder** au moyen de la **méthode factory de Store** :

```
public abstract Folder getFolder(java.lang.String name)  
                                throws MessagingException
```

Folder de base "INBOX".

- Ce folder est fermé au départ, il faut donc l'ouvrir avec la méthode de la classe Folder:

```
public abstract void open(int mode) throws MessagingException
```

Et les variables de classe pour le mode:

```
public static final int READ_ONLY  
public static final int READ_WRITE
```

## 3.3 Programmation de la réception d'un simple message texte: POP3

- On obtient la liste des messages avec:

```
public Message[] getMessages() throws MessagingException
```

Et le nombre de messages stockés :

```
public abstract int getMessageCount() throws MessagingException  
public int getNewMessageCount() throws MessagingException
```

- La liste peut être parcourue afin d'obtenir d'autres renseignements:

```
public abstract Address[] getFrom() throws MessagingException  
public abstract java.lang.String getSubject() throws MessagingException
```

et la méthode de l'interface Part :

```
public java.lang.Object getContent()  
throws java.io.IOException, MessagingException.
```

## 3.3 Programmation de la réception d'un simple message texte: POP3

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

```
public class JMailSimplePartRecv  
{
```

```
    static String host = "u2.tech.hepl.local"; ← Définition du serveur POP3
```

```
    public JMailSimplePartRecv() {}
```

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

```
        Properties prop = System.getProperties();
```

```
        System.out.println("Création d'une session mail");
```

```
        prop.put("mail.pop3.host", host); ← Dans les propriétés, on précise le serveur POP3
```

```
        prop.put("mail.disable.top", true);
```

```
        Session sess = Session.getDefaultInstance(prop, null);
```

← Création d'un objet session

## 3.3 Programmation de la réception d'un simple message texte: POP3

```
try
{
    String user = "vilvens";
    String pwd = "beauGosse";
    Store st = sessgetStore("pop3");
    st.connect(host, user, pwd);
    Folder f = st.getFolder("INBOX");
    f.open(Folder.READ_ONLY);

    Message msg[] = f.getMessages();
    System.out.println("Nombre de messages : " + f.getMessageCount(););
    System.out.println("Nombre de nouveaux messages : " +
        f.getNewMessageCount(););
}
```

1. Pour lire le message, Il faut une instance de Store qui peut être soit IMAP soit POP3 (les providers vont servir). il faut d'abord **instancier un objet Store** en utilisant la `getStore` qui est une factory (voir fichier mail.default.providers qui contient une ligne correspondant à POP3 et une à IMAP. Cela permet de spécifier quelle classe il faut instancier). Dans notre cas, POP3.

2. `connect()` (socket, serveur, pwd/user)

3. Il faut une mailbox = INBOX et l'objet folder ainsi créé voit cette mailbox comme un fichier

4. Ouverture du Folder

5. Demande la liste des messages au Folder

Attention, **getNewMessageCount()** ne fonctionne que pour une seule session. Serait utile uniquement si la session était permanente, ce qui n'est jamais le cas (timeout). Ne pas s'en servir pour compter les nouveaux messages

# 3.3 Programmation de la réception d'un simple message texte: POP3

```
System.out.println("Liste des messages : ");
```

```
for (int i=0; i<msg.length;i++)
```

6. On passe en revue tous les messages

```
{  
  if (msg[i].isMimeType("text/plain"))
```

7. On lit les différentes informations du message

```
{  
    System.out.println("Expéditeur : " + msg[i].getFrom() [0]);  
    System.out.println("Sujet = " + msg[i].getSubject());  
    System.out.println("Texte : " + (String)msg[i].getContent());  
  }  
}
```

8. **getContent** renvoie un Object mais ici on ne s'occupe que de message simple texte, on le « cast » de suite en **String**

Test: if text/Plain , on « cast »  
Méthode: isMimeType()

**Attention**, il faudra toujours faire attention à ce que pourra contenir un message (pas toujours: text/Plain=>String => toujours réaliser la méthode: **getContentType()** de Part

## 3.3 Programmation de la réception d'un simple message texte: POP3

```
catch (NoSuchProviderException e)
{
    System.out.println("Erreur sur provider : " + e.getMessage());
}
catch (MessagingException e)
{
    System.out.println("Erreur sur message : " + e.getMessage());
}
catch (IOException e)
{
    System.out.println("Erreur sur I/O : " + e.getMessage());
}
catch (Exception e)
{
    System.out.println("Erreur indéterminée : " + e.getMessage());
}
}
```

Gestion des exceptions

Problème réseau

Problème: le serveur de messagerie a un problème avec un caractère spécial  
=> message pas vu comme un texte simple. Il faut donc définir les jeux de caractères => préciser:

**Static String charset="iso8859-1"**

**Prop.put("file.encoding", charset)**

**on ajoute dans le properties**

[www.hepl.be](http://www.hepl.be)

## 3.4 Les headers d'un message reçu

Deux champs headers sont utilisés par SMTP. Ces informations permettent de reconstituer le chemin par un message entre le expéditeur et le destinataire (au travers des MTAs):

- **Return-Path:**<address> : adresse de retour en cas de problème
- **Received:**<description du chemin> : champ ajouté par chaque agent intervenant, le chemin suivi pour atteindre cet agent est décrit au moyen des clauses suivantes :

Clause	Signification
from	Site émetteur: (FMTA puis MTA / RMTA puis MDA (MAILSTORE))
by	Site récepteur
via	Chemin physique
with	Protocole utilisé
id	Identificateur du message pour le récepteur
for	Destinataire (si for c'est un FMTA, si on est à destination, plus de for)



## 3.4 Les headers d'un message reçu

- Une méthode de la classe Part fournit une énumération des headers  
Elle fournit une énumération **d'objets Header** (classe du package javax.mail, héritée d'Object), qui ne sont en fait que des paires:  
(<nom de header>, <valeur de ce header>)

```
public java.util Enumeration getAllHeaders() throws MessagingException
```

Avec les méthodes:

```
public java.lang.String getName()  
public java.lang.String getValue()
```

## 3.4 Les headers d'un message reçu

```
...  
for (int i=0; i<msg.length; i++)  
{  
    System.out.println("\n\nHeaders du message n°" + (i+1));  
    Enumeration e = msg[i].getAllHeaders();  
    Header h = (Header)e.nextElement();  
    while (e.hasMoreElements())  
    {  
        System.out.println(h.getName() + " -- >" + h.getValue());  
        h = (Header)e.nextElement();  
    }  
    System.out.println("Texte : " + (String)msg[i].getContent());  
}  
...
```

Pour un éventuel laboratoire: un Jtree pour l'affichage serait judicieux

## 3.4 Les headers d'un message reçu (envoyé à soi-même !)

Vous êtes connecté  
Vous avez: 1 message(s)  
Nombre de messages : 1  
Nombre de nouveaux messages : 0  
Liste des messages :  
Message num : 0  
Expéditeur : chrcharlet@gmail.com  
Sujet = TestMail  
Texte : coucou

Fin des messages

<nHeaders du message n°1

**Return-Path** --> <chrcharlet@gmail.com>

**Received** --> from DESKTOP-BKQ27AG (host-109-89-212-241.dynamic.voo.be. [109.89.212.241])

by smtp.gmail.com with ESMTPSA id rl15-20020a170907216f00b007c0aefd9339sm2594843ejb.175.2023.01.08.07.09.56

for <chrcharlet@gmail.com>

(version=TLS1\_2 cipher=ECDHE-ECDSA-AES128-GCM-SHA256 bits=128/128);

Sun, 08 Jan 2023 07:09:56 -0800 (PST)

Date --> Sun, 8 Jan 2023 16:09:49 +0100 (CET)

**From** --> chrcharlet@gmail.com

To --> chrcharlet@gmail.com

Message-ID --> <1338668845.0.1673190594299@DESKTOP-BKQ27AG>

Subject --> TestMail

MIME-Version --> 1.0

Content-Type --> **text/plain**; charset=us-ascii

Texte : coucou

250 SMTPUTF8  
DEBUG SMTP: Found extension "SIZE", arg "35882577"  
DEBUG SMTP: Found extension "8BITMIME", arg ""  
DEBUG SMTP: Found extension "AUTH", arg "LOGIN PLAIN XOAUTH2 PLAIN-CLIENTTOKEN OAUTHBEARER XOAUTH"  
DEBUG SMTP: Found extension "ENHANCEDSTATUSCODES", arg ""  
DEBUG SMTP: Found extension "PIPELINING", arg ""  
DEBUG SMTP: Found extension "CHUNKING", arg ""  
DEBUG SMTP: Found extension "SMTPUTF8", arg ""  
DEBUG SMTP: protocolConnect login, host=smtp.gmail.com, user=chrcharlet@gmail.com, password=<non-null>  
DEBUG SMTP: Attempt to authenticate using mechanisms: LOGIN PLAIN DIGEST-MD5 NTLM XOAUTH2  
DEBUG SMTP: Using mechanism LOGIN  
DEBUG SMTP: AUTH LOGIN command trace suppressed  
DEBUG SMTP: AUTH LOGIN succeeded  
DEBUG SMTP: use8bit false  
MAIL FROM:<chrcharlet@gmail.com>  
250 2.1.0 OK k22-20020a17090646d600b0085fc3dec567sm1722288ejs.175 - gsmt  
RCPT TO:<chrcharlet@gmail.com>  
250 2.1.5 OK k22-20020a17090646d600b0085fc3dec567sm1722288ejs.175 - gsmt  
DEBUG SMTP: Verified Addresses  
DEBUG SMTP: chrcharlet@gmail.com  
DATA  
354 Go ahead k22-20020a17090646d600b0085fc3dec567sm1722288ejs.175 - gsmt  
Date: Thu, 12 Jan 2023 11:47:40 +0100 (CET)  
From: chrcharlet@gmail.com  
To: chrcharlet@gmail.com  
Message-ID: <1338668845.0.1673520464623@DESKTOP-BKQ27AG>  
Subject: TestMail  
MIME-Version: 1.0  
Content-Type: text/plain; charset=us-ascii  
Content-Transfer-Encoding: 7bit  
  
coucou  
  
.  
250 2.0.0 OK 1673520468 k22-20020a17090646d600b0085fc3dec567sm1722288ejs.175 - gsmt  
DEBUG SMTP: message successfully delivered to mail server  
QUIT  
221 2.0.0 closing connection k22-20020a17090646d600b0085fc3dec567sm1722288ejs.175 - gsmt  
msg envoyé

# 3.4 Les headers d'un message reçu (envoyé à soi-même !)

Configurer le protocole POP dans Gmail



1. Ouvrez [Gmail](#) sur votre ordinateur.

2. En haut à droite, cliquez sur Paramètres

**Afficher tous les paramètres.**

3. Cliquez sur l'onglet **Transfert et POP/IMAP**.

4. Dans la section "Téléchargement POP", sélectionnez **Activer le protocole POP pour tous les messages** ou **Activer le protocole POP pour les messages reçus à partir de maintenant**.

5. Au bas de la page, cliquez sur **Enregistrer les modifications**.

Modifier les paramètres de votre client de messagerie

Accédez à votre client, Microsoft Outlook par exemple, et vérifiez ces paramètres.

Serveur de courrier entrant (POP)

pop.gmail.com  
SSL requis : oui  
Port : 995

Serveur de courrier sortant (SMTP)

smtp.gmail.com  
SSL requis : oui  
TLS requis : oui (si disponible)  
Authentification requise : oui  
Port pour TLS/STARTTLS : 587  
**Si vous utilisez Gmail avec un compte professionnel ou scolaire**, contactez votre [administrateur](#) pour configurer correctement SMTP.

Délai d'expiration du serveur

Doit être supérieur à une minute (5 minutes recommandées)

Nom complet ou nom à afficher

Votre nom

Nom de compte, nom d'utilisateur ou adresse e-mail

Votre adresse e-mail

Mot de passe

Votre mot de passe Gmail

# Création d'un mot de passe d'application

- <https://support.google.com/mail/answer/185833?hl=fr>

## 3.4 Les headers d'un message reçu (de hepl.be vers gmail.com )

<nHeaders du message n°1

**Delivered-To** --> chrcharlet@gmail.com

**Received** --> by 2002:a2e:3803:0:0:0:0:0 with **SMTP** id f3csp3692282lja;

Thu, 12 Jan 2023 02:47:25 -0800 (PST)

**X-Google-Smtp-Source** -->

AMrXdXu1vQVRygctgLWMTRUMBmvfYcP1nE5G1gr7Y2b4dh52Qxf6Gl+lvKGnIjypdB0mtY9VCcj  
w

**X-Received** --> by 2002:a05:6402:185:b0:499:c294:77af with SMTP id r5-

20020a056402018500b00499c29477afmr10391916edv.12.1673520445036;

Thu, 12 Jan 2023 02:47:25 -0800 (PST)

**ARC-Seal** --> i=2; a=rsa-sha256; t=1673520445; cv=pass;

d=google.com; s=arc-20160816;

b=OKbxaXbbSbG7s2ugWvM3qzyGnUJbXfFGqDmlcoxaUHpwcvbeco4ocejY22o1LgSUXm  
7ZbJsZXqRhqU9Uiu41IW9j3YZIF4qOMBDCBXJi3BCV8fWWDTkcD66dfmmt9BTdLmeDn7  
5gz3YhHjgtcDSTCJq+Z4D4FA2phWk9MrMDvpSOVYNJVL00jG87Z4Oxerb+8N/7+3qXbY

dGv69TWk+S+KfExzcmWhMYKm4720Xi9hph4NKvHO1x9FH8qN9JEvOFWcJE2M5hTGps2Y  
VKziu/8VSbFfD1zya63+HID2EqxP0YKJVZWcvRb0tFvQRsURAcANYbgVizCtSSF+z5Ar  
w2Pw==

## 3.4 Les headers d'un message reçu (de hepl.be vers gmail.com )

**ARC-Message-Signature** --> i=2; a=rsa-sha256; c=relaxed/relaxed; d=google.com; s=arc-20160816;  
h=mime-version:msip\_labels:content-language:accept-language  
:message-id:date:thread-index:thread-topic:subject:to:from  
:dkim-signature;  
bh=ppPTOFVh8sHcbuXWySDJ2hIPHsvPsDuyHx/sWGFnQpE=;  
b=w8s0okRs+weMf7Zf430XiwOBQioFgtC3Y OuKM27DFokclRdmuBlSyRGrowpbOjqjwZ  
krHCxFpOheEaMIXHZjiCSDUALkl//QijpThB84/50x5IDIHaPmRTURwzysmFQcQbwvFz  
lj/5MS86WaYz2WEYyC6kjDvwe3gSGQgiFbMY5Xktx52wTeYZrPxgwuJlcJiPpnVOv6ms  
KAceC+vp3wj/zJgu/jf+588nvKSI74FF2QKtqt/57DEwD0TSBbQ6oQ+sjVrgriefMtYe  
jmQF1xZsmAFV0Gmtu0+WxmIDXgktJE8oFt61w7iLCR+7ZnpNkeDWDrST3VN8Di6J7m6N  
GlqQ==

**ARC-Authentication-Results** --> i=2; mx.google.com;  
dkim=pass header.i=@hepl.onmicrosoft.com header.s=selector2-hepl-onmicrosoft-com  
header.b=ulxJc4vo;  
arc=pass (i=1 spf=pass spfdomain=hepl.be dkim=pass dkdomain=hepl.be dmarc=pass  
fromdomain=hepl.be);  
spf=pass (google.com: **domain of christophe.charlet@hepl.be designates 40.107.247.60 as  
permitted sender**) smtp.mailfrom=christophe.charlet@hepl.be



## 3.4 Les headers d'un message reçu (de hepl.be vers gmail.com )

Return-Path --> <christophe.charlet@hepl.be>

Received --> from EUR02-AM0-obe.outbound.protection.outlook.com (mail-am0eur02on2060.outbound.protection.outlook.com. [40.107.247.60])

by mx.google.com with ESMTPS id e23-

20020aa7d7d7000000b00488575d72a1si15852815eds.237.2023.01.12.02.47.24

for <chrcharlet@gmail.com>

(version=TLS1\_2 cipher=ECDHE-ECDSA-AES128-GCM-SHA256 bits=128/128);

Thu, 12 Jan 2023 02:47:25 -0800 (PST)

Received-SPF --> pass (google.com: domain of christophe.charlet@hepl.be designates 40.107.247.60 as permitted sender) client-ip=40.107.247.60;

Authentication-Results --> mx.google.com;

dkim=pass header.i=@hepl.onmicrosoft.com header.s=selector2-hepl-onmicrosoft-com header.b=ulxJc4vo;

arc=pass (i=1 spf=pass spfdomain=hepl.be dkim=pass dkdomain=hepl.be dmarc=pass fromdomain=hepl.be);

spf=pass (google.com: domain of christophe.charlet@hepl.be designates 40.107.247.60 as permitted sender) smtp.mailfrom=christophe.charlet@hepl.be

ARC-Seal --> i=1; a=rsa-sha256; s=arcselector9901; d=microsoft.com; cv=none;

b=Es3tRL16r0P1qR8mo/tTOu9RUPi9C7Fx6buUktB52OHq39F9I73CjmgkpGaBYN3UIDpM5CEmQXJFQ  
ToXu9jRdmQ1+vYvzXfTsINDGqWaNpC6Fov2Ox0T5cTrbvDyIXBUxMNRxzJjqLOz1xcO2f+cDNUX0qqTX  
3FROqwJmKnXPesdTPmwDQtx9Jv/XLJ3+hOoEJnTahqdcM98cDukpNVZGsGz/XV/skCP3OKSzDnK6H  
4qRGc7V2Zwfygyf4f6rVuD7d9vyCdICWLjDPnR4/3VQ7xo3BhwHgp7c7qIDBdvJJ/pik371QXJOEKRI5Z6q  
vqetNuWPRZDrhKmecleU9+ePw==

## 3.4 Les headers d'un message reçu (de hepl.be vers gmail.com )

ARC-Message-Signature --> i=1; a=rsa-sha256; c=relaxed/relaxed; d=microsoft.com;  
s=arcselector9901;  
h=From:Date:Subject:Message-ID:Content-Type:MIME-Version:X-MS-Exchange-AntiSpam-  
MessageData-ChunkCount:X-MS-Exchange-AntiSpam-MessageData-0:X-MS-Exchange-  
AntiSpam-MessageData-1;  
bh=ppPTOFVh8sHcbuXWYSDJ2hIPHsvPsDuyHx/sWGFNqPE=;

b=ewnfncwcmxmV5xIZOCUp7ZpuXJ/dIDP7Qjr0TuDLefQeSONWI8m0FZ9PjIHarcqCTZSBqd7wdU9  
+EACBbRNo65fg6Z8aTFTLHYI7ujO6R2qSwBgRgHDSJI/A8dIPt+P1861zLLESb6KzrJWcLivUIF/Q  
xdoBv+WptalOOMRaeP1aSB4a0QTsNAeFDWIFKxC/WRg5bcPiCkCu4+2rG8zPyXX+TG5tQBEInI  
DbCY5owDmQ2fgG+Hsih1o8HPhIKnHAIK1nUkvG9qOMRFCMWEDsLrWbRa1/r8/Bk+noCaGMhjj  
N/PB3FxSiRIccorOz4tjsPnQvDk4rvYo/rF8iXXOKIM2g==

ARC-Authentication-Results --> i=1; mx.microsoft.com 1; spf=pass  
smtp.mailfrom=hepl.be; dmarc=pass action=none header.from=hepl.be; dkim=pass  
header.d=hepl.be; arc=none

DKIM-Signature --> v=1; a=rsa-sha256; c=relaxed/relaxed; d=hepl.onmicrosoft.com;  
s=selector2-hepl-onmicrosoft-com;  
h=From:Date:Subject:Message-ID:Content-Type:MIME-Version:X-MS-Exchange-  
SenderADCheck;  
bh=ppPTOFVh8sHcbuXWYSDJ2hIPHsvPsDuyHx/sWGFNqPE=;

b=ulxJc4vo/Cfy4dHqqOEcpf9msaEvhFCkgmkv1kGNqQ7Bqt2OKv/I+xwmB/ID20YI357zk8Hu03rrL  
mZcKFuWcfWqfpTeh1IOXv2yXxVpXefmN/0t3GMEjYNSM9czwVzN0Otsu5IYANjtduKPVeCj6giPX  
DnTPDP82yLWUz8PYN8=

## 3.4 Les headers d'un message reçu (de hepl.be vers gmail.com )

Received --> from PA4PR01MB8922.eurprd01.prod.exchangelabs.com

(2603:10a6:102:2a2::9) by AS4PR01MB10060.eurprd01.prod.exchangelabs.com  
(2603:10a6:20b:4f6::8) with Microsoft SMTP Server (version=TLS1\_2,  
cipher=TLS\_ECDHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384) id 15.20.6002.13; Thu, 12 Jan  
2023 10:47:24 +0000

Received --> from PA4PR01MB8922.eurprd01.prod.exchangelabs.com

([fe80::cce1:c118:6dd:dc52]) by PA4PR01MB8922.eurprd01.prod.exchangelabs.com  
([fe80::cce1:c118:6dd:dc52%5]) with mapi id 15.20.6002.013; Thu, 12 Jan 2023  
10:47:24 +0000

From --> CHARLET CHRISTOPHE <christophe.charlet@hepl.be>

To --> "chrcharlet@gmail.com" <chrcharlet@gmail.com>

Subject --> Pour les Headers

Thread-Topic --> Pour les Headers

Thread-Index --> AQHZJnNA+0C31qtdUUucwVtxe/d9YQ==

Date --> Thu, 12 Jan 2023 10:47:23 +0000

Message-ID -->

<PA4PR01MB892203B01A9A2055689243158CFD9@PA4PR01MB8922.eurprd01.prod.exchangelab  
s.com>

Accept-Language --> fr-BE, en-US

Content-Language --> fr-BE

# 3.4 Les headers d'un message reçu (de hepl.be vers gmail.com )

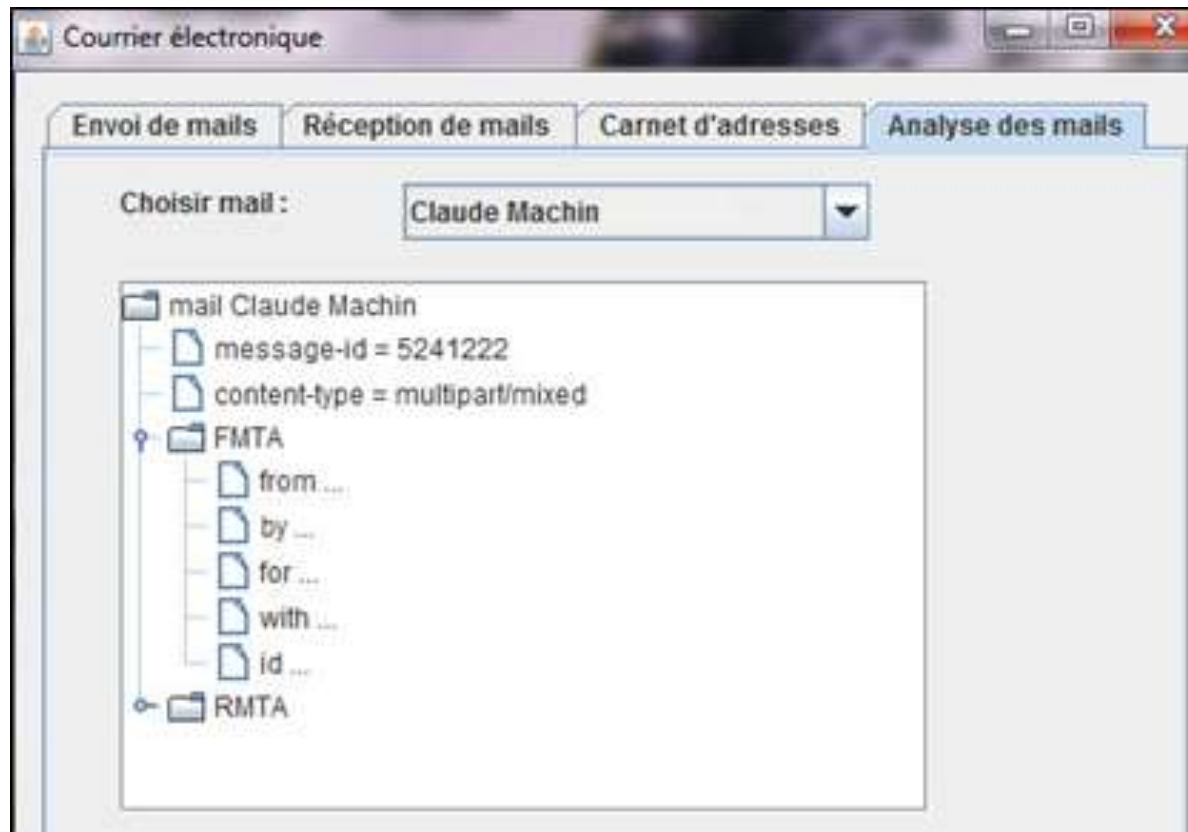
.....  
=?iso-8859-1?Q?QHdz62cHRE1/AyAa4JlIF7ujjzq4VQE1yh5QW1OrQFrJseklyy6Qs6MrzX?=  
=?iso-8859-1?Q?4go5OY4fBETsNKjiuPxLMaCAmVNomp2dX/0bK3Vw?=  
Content-Type --> multipart/alternative;  
boundary="\_000\_PA4PR01MB892203B01A9A2055689243158CFD9PA4PR01MB8922e  
urp\_"  
MIME-Version --> 1.0  
X-OriginatorOrg --> hepl.be  
X-MS-Exchange-CrossTenant-AuthAs --> Internal  
X-MS-Exchange-CrossTenant-AuthSource --> PA4PR01MB8922.eurprd01.prod.exchangelabs.com  
X-MS-Exchange-CrossTenant-Network-Message-Id --> 801381f1-d68f-44a9-2b44-08daf48a639c  
X-MS-Exchange-CrossTenant-originalarrivaltime --> 12 Jan 2023 10:47:23.9721  
(UTC)  
X-MS-Exchange-CrossTenant-fromentityheader --> Hosted  
X-MS-Exchange-CrossTenant-id --> 40f5a870-cf58-4663-9ebd-dab2becb898d  
X-MS-Exchange-CrossTenant-mailboxtype --> HOSTED  
X-MS-Exchange-CrossTenant-userprincipalname -->  
Bxz4gxXHRWZKW8WWfzVBDOC4S9n7GnlZ8nKtb5701zdOStnQP9c+knIWP3hlj6ba2Dh/6ePL5uK1z  
q/NhNCtAaS/AzfPzXLGIv0/Z50QHTY=  
Erreur indéterminée : javax.mail.internet.MimeMultipart cannot be cast to java.lang.String

Curieux, une idée ?



## 3.4 Les headers d'un message reçu

Pour un éventuel laboratoire: un Jtree pour l'affichage serait judicieux



## **4. MESSAGE AVEC PIÈCE(S) ATTACHÉE(S)**

# 4.1 Message avec pièce(s) attachée(s)

## Exemple d'envoi avec 3 Parts

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

public class JMailMultiplePart
{
    static String host = "u2.tech.hepl.local";
    static String charset = "iso-8859-1";
    public JMailMultiplePart() {}

    public static void main (String args[])
    {
        Properties prop = System.getProperties();
        prop.put("mail.smtp.host", host);
        prop.put("file.encoding", charset);
        System.out.println("Création d'une session mail");
        Session sess = Session.getDefaultInstance(prop, null);
```

3 Parts:

- Simple texte (texte du message)
- Fichier .doc
- Image .bmp

Récupération des propriétés

Ajout dans les propriétés de serveur

Définition du charset dans les propriétés

Création de l'objet session à partir des propriétés

# 4.1 Message avec pièce(s) attachée(s)

## Exemple d'envoi avec 3 Parts

try

{

```
String exp = "vilvens@u2.tech.hepl.local";
```

```
String dest = "charlet@u2.tech.hepl.local";
```

```
String sujet = "Essai d'attachement";
```

```
String texteAcc = "Veuillez trouver ci-joint les documents demandés - CV";
```

Référence de la session qui contient notamment les infos sur le serveur SMTP

```
MimeMessage msg = new MimeMessage (sess);
```

Instanciation du MimeMessage()

```
msg.setFrom (new InternetAddress (exp));
```

```
msg.setRecipient (Message.RecipientType.TO, new InternetAddress (dest));
```

```
msg.setSubject(sujet);
```

Idem msg texte simple

1ère composante : le texte d'accompagnement

```
System.out.println("Début construction du multipart");
```

```
Multipart msgMP = new MimeMultipart();
```

```
System.out.println("1ère composante");
```

```
MimeBodyPart msgBP = new MimeBodyPart();
```

```
msgBP.setText(texteAcc);
```

```
msgMP.addBodyPart(msgBP);
```

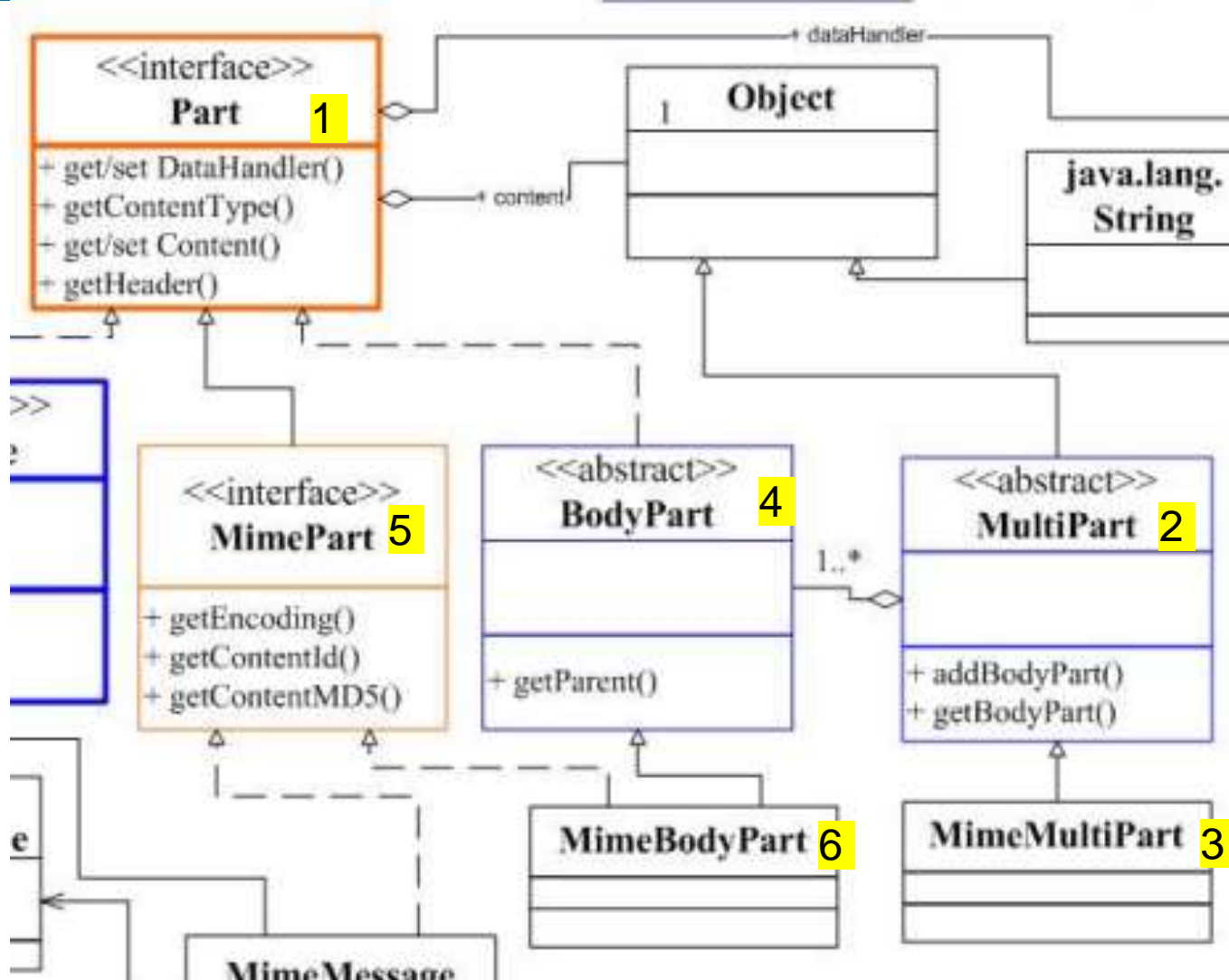
= Un conteneur de Part() qui contient des objets de type MimeBodyPart() dérivée de BodyPart() (abstract et implémente l'interface Part) « chargé » grâce à la méthode addBodyPart()

Méthode setText sur l'objet MimeBodyPart car simple texte



# 4.1 Message avec pièce(s) attachée(s)

## Exemple d'envoi avec 3 Parts



# 4.1 Message avec pièce(s) attachée(s)

## Exemple d'envoi avec 3 Pièces

2ème composante : le fichier Word

```
System.out.println("2ème composante");  
String nf = "d:\\notes-java\\BienvenueAInpres.doc";  
msgBP = new MimeBodyPart();  
DataSource so = new FileDataSource(nf);  
msgBP.setDataHandler(new DataHandler(so));  
msgBP.setFileName(nf);  
msgMP.addBodyPart(msgBP);
```

On passe par un DataHandler() qui est une classe qui sait comment manipuler les sources de données et qui compose le contenu

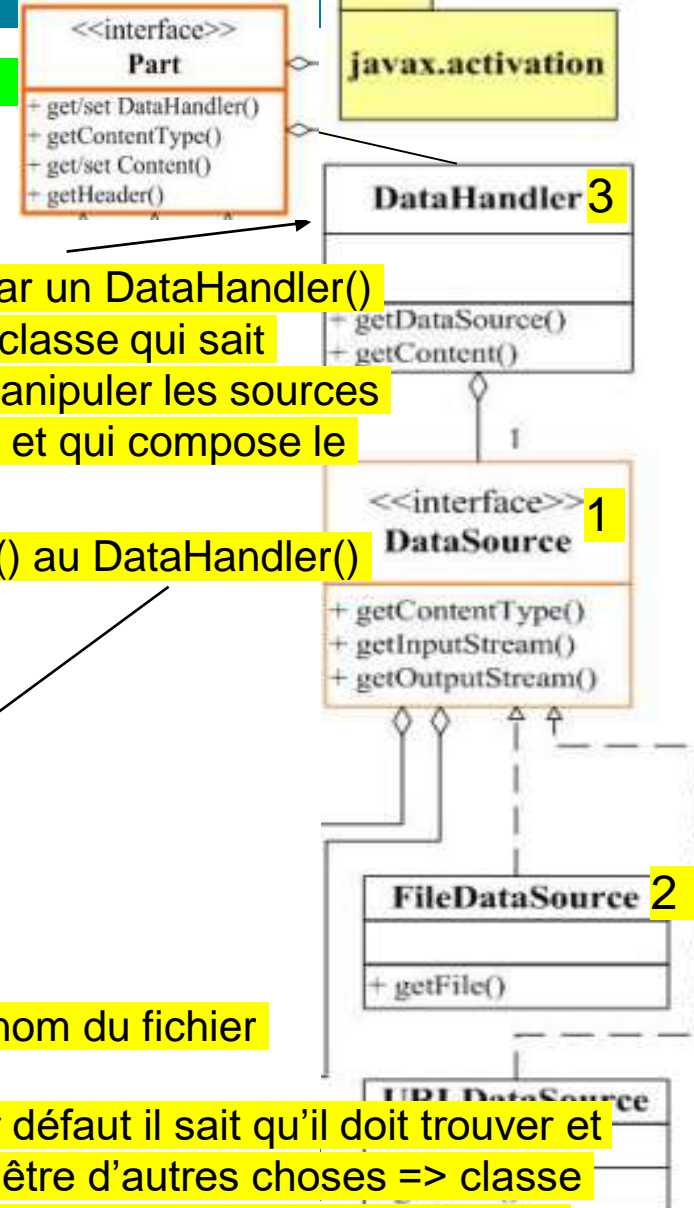
On passe le DataSource() au DataHandler()

3ème composante : l'image

```
System.out.println("3ème composante");  
nf = "d:\\notes-java\\logo-INPRES.bmp";  
msgBP = new MimeBodyPart();  
so = new FileDataSource(nf);  
msgBP.setDataHandler(new DataHandler(so));  
msgBP.setFileName(nf);  
msgMP.addBodyPart(msgBP);
```

On ajoute le nom du fichier

Le DataHanler() est une classe instanciable: par défaut il sait qu'il doit trouver et écrire les données dans un fichier. Mais ça peut être d'autres choses => classe dérivée de DataHandler() données dans un serveur ou sur une carte à puce.



## 4.1 Message avec pièce(s) attachée(s) Exemple d'envoi avec 3 Parts

```
msg.setContent(msgMP);
```

Le contenu du multipart est le contenu du message => Le message est un multipart (les 3 morceaux sont dans le multipart)

```
System.out.println("Envoi du message");  
Transport.send(msg);
```

```
System.out.println("Message envoyé");  
}  
catch (AddressException e)  
{  
    System.out.println("Erreur sur message : " + e.getMessage());  
}  
catch (MessagingException e)  
{  
    System.out.println("Erreur sur message : " + e.getMessage());  
}  
}  
}
```

## 4.2 Classes pour les messages avec pièce(s) attachée(s)

L'interface **DataSource** (du package **javax.activation**) représente une collection quelconque de données qui seront accédées comme des **InputStreams** et **OutputStreams**.

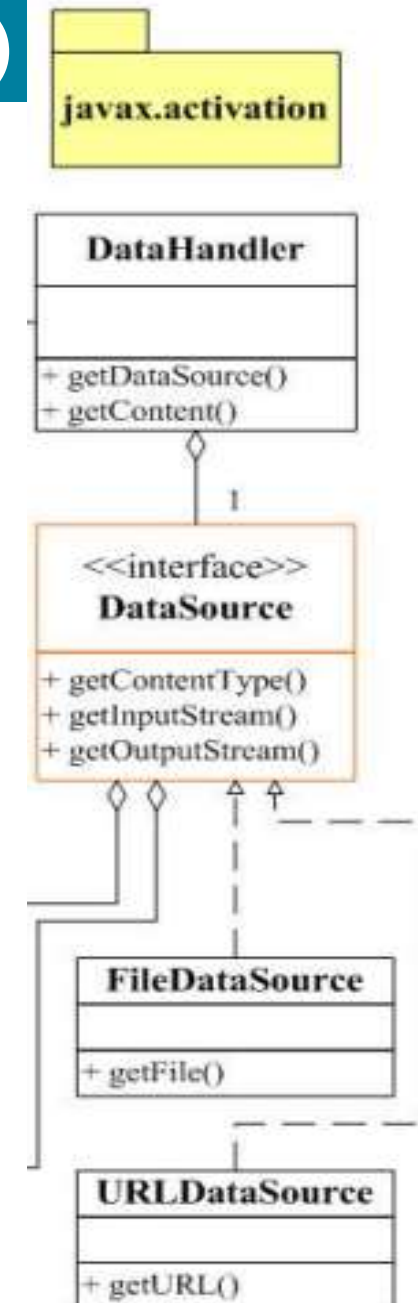
Les méthodes sont les suivantes:

**public java.lang.String getContentType()**  
qui renvoie le type MIME de la donnée

**public java.lang.String getName()**  
qui renvoie le nom de l'objet qui représente la donnée

**public java.io.InputStream getInputStream()**  
throws java.io.IOException  
**public java.io.OutputStream getOutputStream()**  
throws java.io.IOException

qui fournit les flux représentant les données en entrée ou en sortie, si c'est possible (une exception est lancée dans le cas contraire).



## 4.2 Classes pour les messages avec pièce(s) attachée(s)

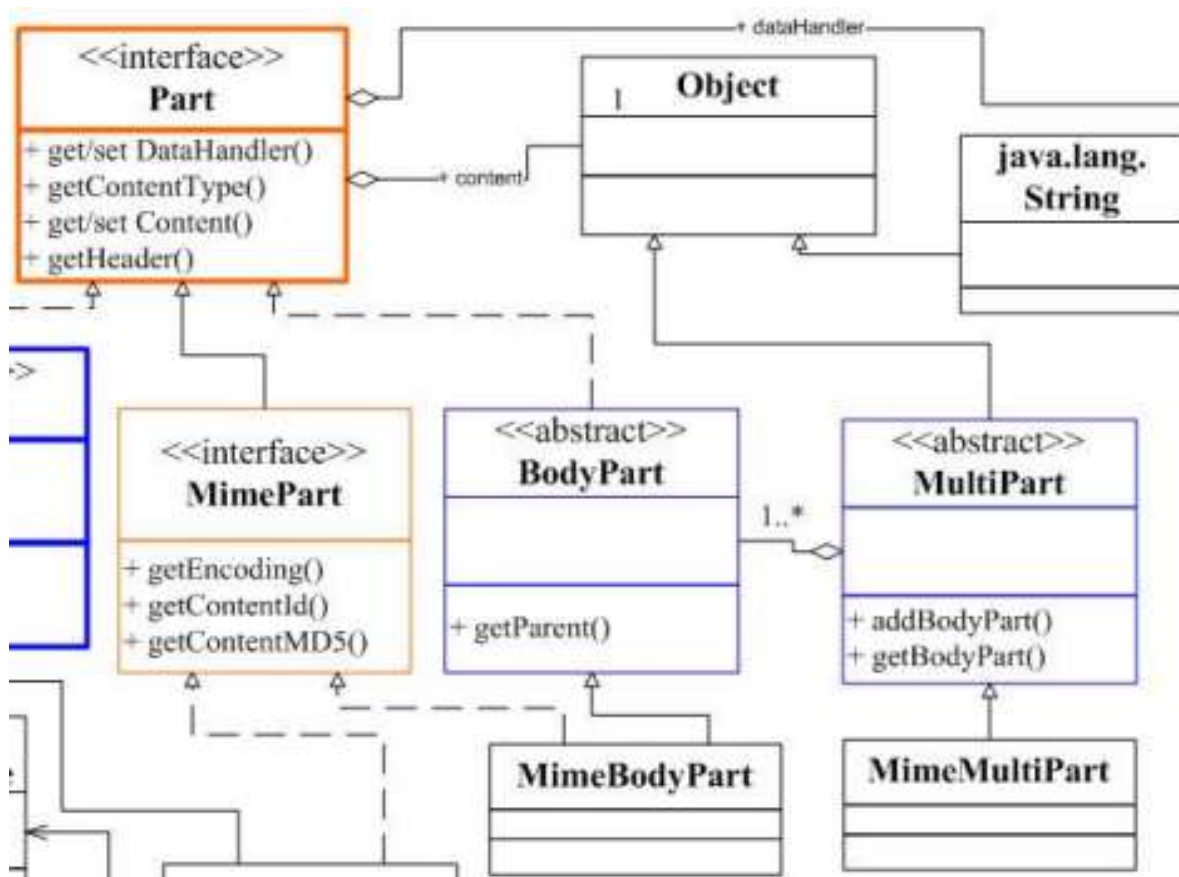
### Les messages avec pièces attachées (**Multipart**)

Header	
	Attributs définis dans Part, dont <b>Content-Type</b> vaut <b>Multipart</b>
	Attributs ajoutés par Message, dont From, To, Subject
	Attributs ajoutés par MimeMessage, dont les flags
Corps [ <i>Content body</i> ]	
	Objet DataHandler donnant accès à un objet Multipart

## 4.2 Classes pour les messages avec pièce(s) attachée(s)

Classe abstraite **Multipart** dérivée d'**Objet** dont le type MIME est par défaut multipart/mixed, est un container d'objets *BodyPart*

Variable membre: **protected java.util.Vector parts**



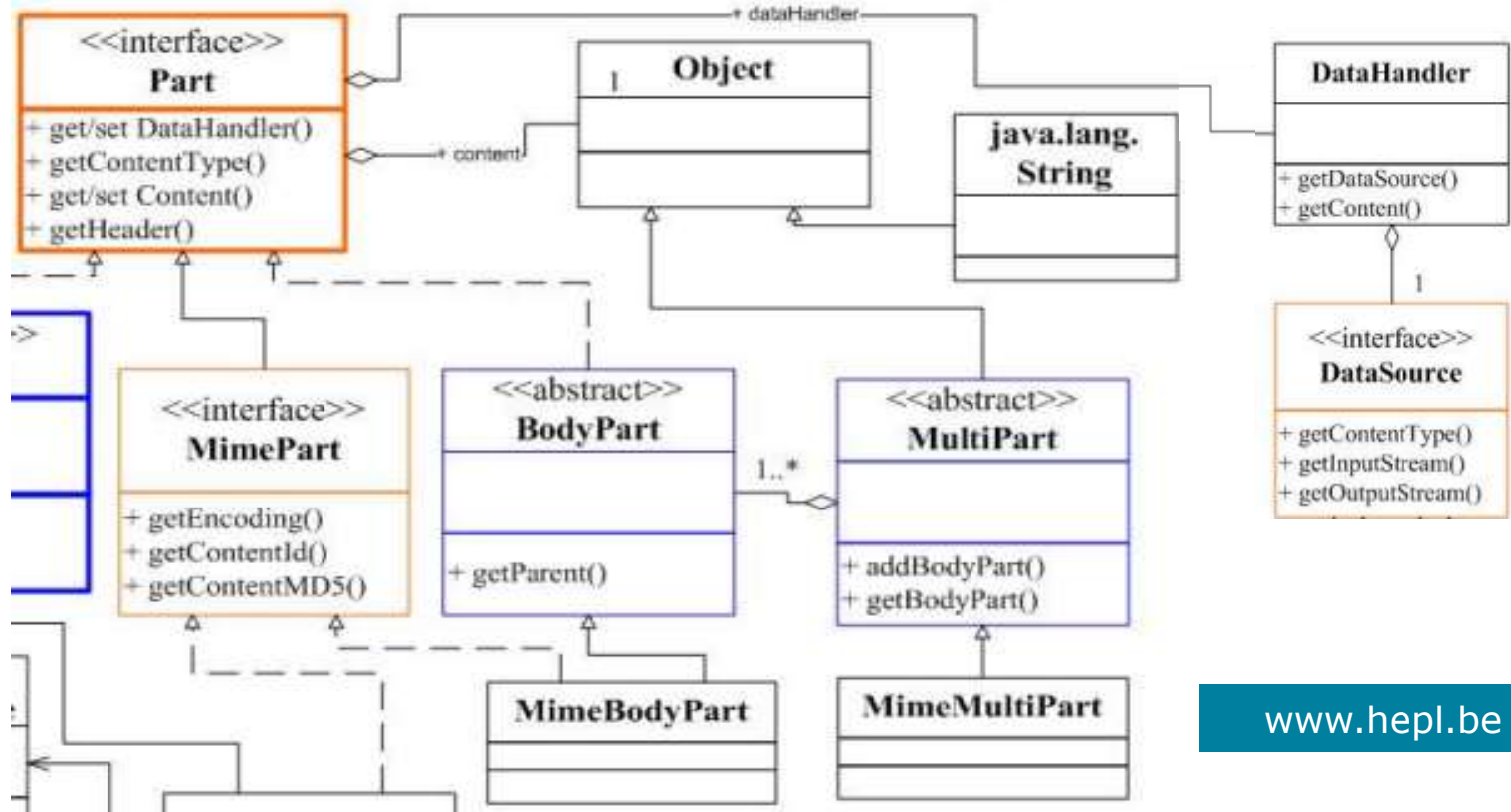
## 4.2 Classes les messages avec pièce(s) attachée(s)

La classe abstraite **BodyPart** implémente l'interface Part

Chaque objet BodyPart peut donner accès, par son DataHandler à :

- des données simples
- un autre objet Multipart

On peut donc ainsi créer des hiérarchies. la variable membre de Multipart : **protected Part parent**



## 4.2 Classes pour les messages avec pièce(s) attachée(s)

Header	
	Attributs définis dans Part, dont <b>Content-Type</b> vaut <b>Multipart</b>
	Attributs ajoutés par Message, dont From, To, Subject
	Attributs ajoutés par MimeMessage, dont les flags
Corps [ <i>Content body</i> ]	
	Objet DataHandler <b>donnant accès à un objet Multipart</b>

objet Multipart	
	<b>Objet BodyPart 1</b>
	<b>Header</b>
	Attributs définis dans Part
	<b>Corps</b>
	Objet DataHandler donnant accès au contenu
	<b>Objet BodyPart 2</b>
	<b>Header</b>
	Attributs définis dans Part, dont <b>Content-Type</b> vaut <b>Multipart</b>
	<b>Corps</b>
	Objet DataHandler <b>donnant accès à un objet Multipart</b>



## 4.2 Classes les messages avec pièce(s) attachée(s)

**la méthode de Part :**

`public java.lang.String getDisposition() throws MessagingException`

Permet de savoir si on a: une pièce attachée ("ATTACHMENT")  
une donnée distante ("INLINE"),

Pour le texte accompagnateur, on teste à l'aide de :

`isMimeType("text/plain")`

ce que l'on teste en utilisant les deux constantes de classe :

`public static final java.lang.String ATTACHMENT`

`public static final java.lang.String INLINE`

**Création d'un flux de lecture**

`public java.io.InputStream getInputStream()`

`throws java.io.IOException, MessagingException`

Vu le nombre bytes inconnu: `ByteArrayOutputStream`

**Récupération du nom de la pièce attachée**

`public java.lang.String getFileName() throws MessagingException`

## 4.3 Message avec pièce(s) attachée(s)

### Exemple de réception avec 3 Parts

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

public class JMailMultiplePartRecv
{

    static String host = "u2.tech.hepl.local";
    public static void main (String args[])

    {

        Properties prop = System.getProperties();
        System.out.println("Création d'une session mail");
        Session sess = Session.getDefaultInstance(prop, null);
```

## 4.3 Message avec pièce(s) attachée(s)

### Exemple de réception avec 3 Parts

```
try
```

```
{
```

```
String user = "xyz54321";
```

```
String pwd = "abc123";
```

```
System.out.println("Obtention d'un objet store");
```

```
Store st = sessgetStore("pop3");
```

```
st.connect(host, user, pwd);
```

```
System.out.println("Obtention d'un objet folder");
```

```
Folder f = st.getFolder("INBOX");
```

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

```
System.out.println("Obtention des messages");
```

```
Message msg[] = f.getMessages();
```

```
System.out.println("Nombre de messages : " + f.getMessageCount());
```

```
System.out.println("Nombre de nouveaux messages : " + f.getNewMessageCount());
```

Cherche une classe qui implémente la classe abstraite Store (elle se trouve dans le fichier `javaMail.default.Provider`: on trouve la ligne correspondant à l'instanciation de la classe (POP3) pour obtenir un objet Store

Ouverture de la mailbox

Lecture de tous les messages

## 4.3 Message avec pièce(s) attachée(s)

### Exemple de réception avec 3 Parts

```
System.out.println("Obtention des messages");
```

```
Message msg[] = f.getMessage();
```

```
System.out.println("Nombre de messages : " + f.getMessageCount());
```

```
System.out.println("Nombre de nouveaux messages : " + f.getNewMessageCount());
```

Remarque: ne prendre que les messages qui concernent le labo (le reste ne regarde personne):

```
System.out.println("Liste des messages : ");
```

```
for (int i=0; i<msg.length; i++)  
{  
    //String[] rt=msg[i].getHeader("Return-Path");  
    //String[] rt=msg[i].getHeader("From");  
    String[] rt=msg[i].getHeader("Subject");  
    System.out.println("coucou " + rt[0] + " coucou");  
    pos = rt[0].indexOf("TestMail");  
    System.out.println("coucou " + pos + " coucou");  
}
```

Exemple en utilisant différents headers  
afin de sélectionner des mails dans la  
liste des message (msg[])

## 4.3 Message avec pièce(s) attachée(s)

### Exemple de réception avec 3 Parts

```
System.out.println("Liste des messages : ");
```

```
for (int i=0; i<msg.length; i++)  
{
```

```
    System.out.println("Message n° " + i);
```

```
    System.out.println("Expéditeur : " + msg[i].getFrom() [0]);
```

```
    System.out.println("Sujet = " + msg[i].getSubject());
```

```
    System.out.println("Date : " + msg[i].getSentDate());
```

```
// Récupération du conteneur Multipart
```

```
if (*****)  
{ ....
```

Attention: il faudrait d'abord tester si on a affaire effectivement à un multipart => un if avec un `isMimeType()`

```
Multipart msgMP = (Multipart)msg[i].getContent();
```

```
int np = msgMP.getCount();
```

```
System.out.println("-- Nombre de composantes = " + np);
```

Casté d'office en multipart (pour Outlook, c'est par default un multiPart (2 parties msg en clair et en HTML))

## 4.3 Message avec pièce(s) attachée(s)

### Exemple de réception avec 3 Parts

```
for (int j=0; j<np; j++)
{
    System.out.println("--Composante n° " + j);
    Part p = msgMP.getBodyPart(j);
    String d = p.getDisposition();
    if (p.isMimeType("text/plain"))
    System.out.println("Texte : " + (String)p.getContent());
    if (d!=null && d.equalsIgnoreCase(Part.ATTACHMENT))
    {
        InputStream is = p.getInputStream();
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        int c;
        while ((c = is.read()) != -1) baos.write(c);
        baos.flush();
        String nf = p.getFileName();
        FileOutputStream fos = new FileOutputStream(nf);
        baos.writeTo(fos); fos.close();
        System.out.println("Pièce attachée " + nf + " récupérée");
    }
}
System.out.println("Fin des messages"); }
```

**Scan des BodyPart**

On récupère aussi « la disposition ». On récupère la constante « ATTACHMENT » ce qui signifie que c'est une vraie pièce attachée (=> fichier) si ONLINE = pièce attachée sur un serveur WEB (=> classe URL connexion)

**P est le Part**

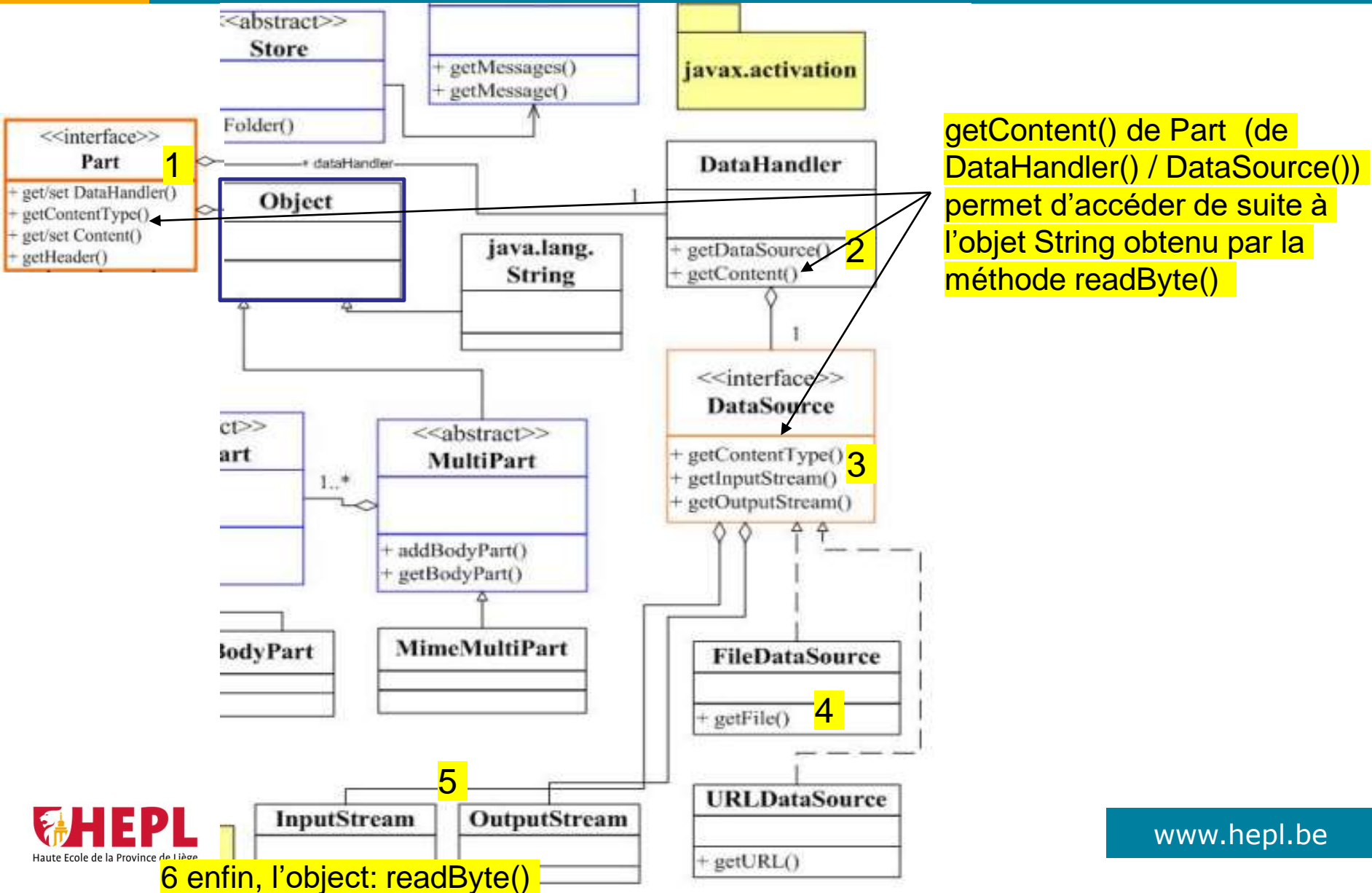
On teste dans cette boucle le type des différents Parts: Text/Plain ou autre chose (fichier dans notre cas). Il faudrait plusieurs if pour les différents types possibles

On récupère les bytes de la pièce attachée dans un ByteArrayOutputStream() (= flux mémoire, pas de dimension car taille inconnue)

On écrit le contenu de la mémoire (baos) dans un fichier, le getFileName() récupère le nom qu'on avait stocké à l'envoi

## 4.3 Message avec pièce(s) attachée(s)

### Exemple de réception avec 3 Parts



# 5. IMAP (INTERNET MESSAGE ACCESS PROTOCOL)



# 5.1 IMAP: Présentation

- **POP3** permet aux utilisateurs connectés seulement par intermittence, de télécharger leurs mails sur leur machine locale à chaque connexion, les mails pouvant être détruits ou conservés sur le serveur (dans les limites de l'espace autorisé).
- **IMAP:**
  - Permet: aux utilisateurs d'accéder comme si elle était locale (sans download immédiat).
  - Les messages peuvent être consultés par plusieurs clients simultanément
  - Les folders peuvent être partagés.

IMAP	
Nom complet:	Internet Message Access Protocol
Nature:	Protocole applicatif de récupération des messages électroniques
RFC	2060, 1733 + 2086
Port par défaut	143 (non chiffré) / Port 993 – Port SSL / TLS
Protocole de transport	TCP
Principales commandes	login, select, search, fetch, store, expunge, logout

# 5.1 IMAP: Présentation

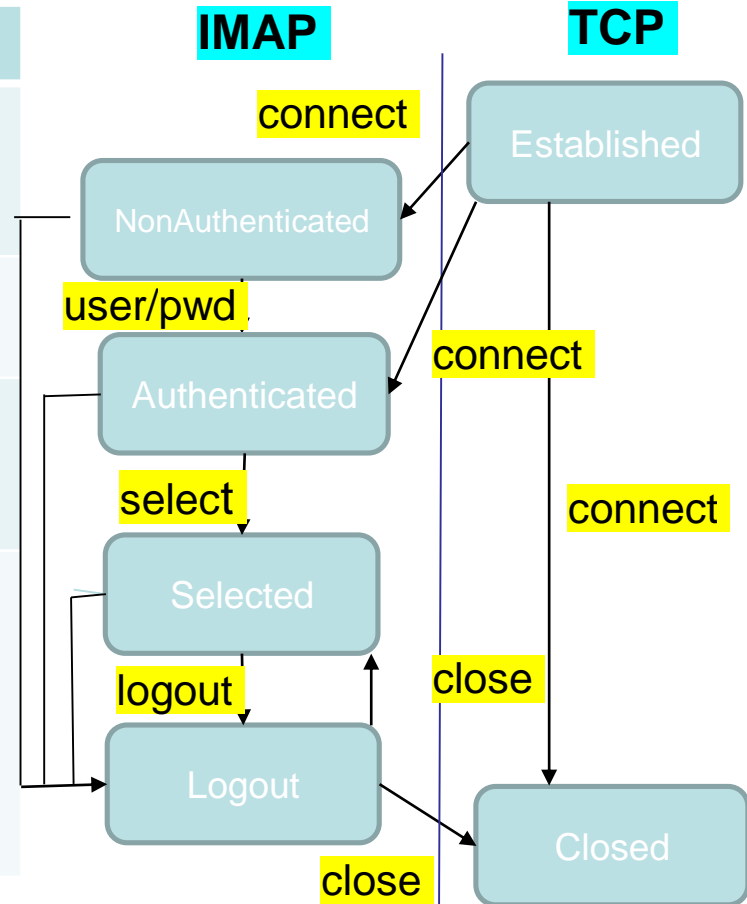
- **IMAP:**

- Fonctionne comme un démon qui attend des connexions sur son port.
- Un client se connecte sur **le port d'écoute** (IMAP), ensuite un **sous processus (ou thread)** est créé et prend en charge sur un autre port et attend les commandes d'accès à la boîte aux lettres visée.
- un client IMAP est sensé être un **MUA multithread** : il peut envoyer plusieurs commandes simultanément et recevoir une réponse d'un serveur à tout moment, même s'il est occupé simultanément à autre chose => **plusieurs communications simultanées**.
- **Pour identifier un couple request/response**, le client ajoute à la commande une valeur alphanumérique du type par exemple : client ->TAG25 et serveur -> TAG 25
- C'est un **protocole à états** comme POP3.
- Possède de nombreuses commandes, paramétrables au moyen de nombreux commutateurs:extraction d'une partie bien précise d'un message, la lecture et la modification des flags des messages: message récent ou pas, lu ou pas, doit être effacé, déjà répondu ou pas + la gestion des mailbox, ....

# 5.1 IMAP: Présentation

**IMAP un protocole à états (4)** car contrairement à POP3, il peut y avoir plusieurs mailbox => sélectionner la mailbox

Etats	Signification
NON_AUTHENTICATED	La connexion TCP étant acceptée, le client doit à présent s'authentifier
AUTHENTICATED	Le client s'est connecté et fait reconnaître par le serveur.
SELECTED	Le client a sélectionné la boîte aux lettres sur laquelle il va effectuer diverses opérations.
LOGOUT	Dès que la connexion au serveur est perdue, soit parce que le client a terminé, soit parce que le serveur refuse ses services, ou encore parce que la connexion a été coupée.



## 5.2 IMAP: Les commandes de base (consultation des messages)

Commandes	Etats valides	Description
<b>Login</b> <nom utilisateur><pwd>	NON AUTHENTICATED	Permet de passer à l'état authentifié
<b>Select</b> <nom de la mailbox	AUTHENTICATED ou SELECTED	Permet de sélectionner une boîte aux lettres – par exemple : INBOX.
<b>Search</b> <all   subject<ch>[not] body<ch>>	SELECTED	Permet de rechercher les messages correspondant à tel ou tel critère (la liste des flags est assez longue); on obtient ainsi les numéros des messages (uid) se trouvant dans la mailbox
<b>Fetch</b> <num de msg><all   body   body [header   text   mime] enveloppe   uid rfc822>	SELECTED	Permet de lire le message du numéro spécifié, en précisant la partie du mail souhaité (la liste des flags est assez longue).
expunge	SELECTED	Supprime les messages dont le flag "deleted" est positionné, ce qui se fait avec la commande suivante

## 5.2 IMAP: Les commandes de base (consultation des messages)

Commandes	Etats valides	Description
<b>Store</b> <num msg>+FLAGS   -FLAGS (\answered   \deleted   \seen)	SLECTED	Permet de modifier le flag précisé en fin de commande pour le message du numéro précisé.
<b>logout</b>	tous	
<b>noop</b>	tous	L'opération vide – elle sert à montrer que le client est toujours vivant ...
<b>capability</b>	tous	Renseigne sur les disponibilités du serveur.

Un message est désigné au moyen :

- d'un numéro de message au sein de sa mailbox; il peut être modifié car si on efface un message, les numéros des messages suivants sont décrémentés;
- d'un UID (Unique IDentifier) qui est attribué définitivement au message à son arrivée dans la mailbox; il ne sera jamais réutilisé dans la mailbox sélectionnée; par contre, des messages se trouvant dans des mailbox différentes peuvent avoir le même UID.

## 5.2 IMAP: Les commandes de base (consultation des messages)

- 4 réponses de base aux différentes commandes:
  - OK,
  - BAD (erreur dans la syntaxe de la cmd) ,
  - NO (erreur dans la réalisation de la cmd),
  - BYE (fermeture de connexion)
- Les réponses sont taggées (identification du couple cmd/réponse)
- Le serveur peut fournir des informations supplémentaires utiles. C'est dernières ne sont pas taggées mais précédées de \* .
- Codes d'erreur supplémentaires:
  - ALERT,
  - FLAGS,
  - PERMANET-FLAGS (flags qui peuvent être modifiés en permanence)
  - UNSEEN (num du 1er msg non lu)
  - PARSE (le header du message ne peut être analysé correctement)

## 5.2 IMAP: Les commandes de base (consultation des messages)

vilvens@cure:~ > telnet u2 143

Trying 10.59.5.219...

Connected to u2.

Escape character is '^['.

OK [CAPABILITY IMAP4 IMAP4REV1 STARTTLS LOGIN-REFERRALS AUTH=LOGIN] u2.wildne)

tag1 capability

CAPABILITY IMAP4 IMAP4REV1 STARTTLS NAMESPACE IDLE MAILBOXREFERRALS SCAN SORN

tag1 OK CAPABILITY completed

tag2 login vilvens Etpuisquoiencore

\* CAPABILITY IMAP4 IMAP4REV1 STARTTLS NAMESPACE IDLE MAILBOXREFERRALS SCAN SORD

tag2 OK LOGIN completed

tag3 select inbox

\* 2 EXISTS

\* 2 RECENT

\* OK [UIDVALIDITY 1031664763] UID validity status

\* OK [UIDNEXT 3] Predicted next UID

\* FLAGS (\Answered \Flagged \Deleted \Draft \Seen)

\* OK [PERMANENTFLAGS (\\* \Answered \Flagged \Deleted \Draft \Seen)] Permanent fs

\* OK [UNSEEN 1] first unseen message in /var/spool/mail/vilvens

tag3 OK [READ-WRITE] SELECT completed

tag4 search all \* SEARCH 1 2

tag4 OK SEARCH completed

tag5 fetch 2 all \* 2 FETCH (FLAGS (\Recent) INTERNALDATE "10-Sep-2002 15:26:27 +0200"

RFC822.SIZ)

tag5 OK FETCH completed

## 5.2 IMAP: Les commandes de base (consultation des messages)

```
tag6 fetch 2 body[header]
2 FETCH (BODY[HEADER] {397} X-UIDL: +>2!!><O!!7+3"!-j="!
Return-Path: <vilvens@u2.wildness.loc>
Received: from ulyse (vilvens@[10.59.5.224]) by u2.wildness.loc (8.11.3/jtpda-5.3.3) with SMTP id
g8ADOrS25724
for ; Tue, 10 Sep 15:25:12 +0200
Date: Tue, 10 Sep 15:25:12 +0200
From: vilvens@u2.wildness.loc
Message-Id: <200209101325.g8ADOrS25724@u2.wildness.loc>
```

Indique par le message est passé



## 5.3 IMAP: Les commandes de base (Manipulation de la mailbox)

Commandes	Etats valides	Description
<b>create</b> <nom de la mailbox>	AUTHENTICATED ou SELECTED	Permet de créer une nouvelle boîte aux lettres.
<b>List</b> <nom folder><nom de la mailbox>	AUTHENTICATED ou SELECTED	Fournit la liste des boîtes aux lettres accessibles à l'utilisateur – les caractères jokers '*' et '%' peuvent être utilisés.
<b>Rename</b> <nom actuel de la mail box> <nouveau nom de la mailbox>	AUTHENTICATED ou SELECTED	Permet de modifier le nom d'une boîte aux lettres
<b>close</b>	SELECTED	Ferme la boîte aux lettres en cours d'utilisation et valide l'effacement des messages marqués pour la destruction – le client revient à l'état AUTHENTICATED.
<b>Delete</b> <nom de la mailbox>	AUTHENTICATED ou SELECTED	Evident !
<b>Append</b> <nom de la mailbox> {<nbr caractères>}	AUTHENTICATED ou SELECTED	Placera le message qui va être défini par après à la fin de la boîte aux lettres spécifiée.

# 5.4 IMAP: en JAVA

## (Réception simple mail texte)

```
public class JMailSimplePartRecvImap
{
    static String host = "u2.wildness.loc";
    public JMailSimplePartRecvImap() { }
    public static void main (String args[])
    {
        Properties prop = System.getProperties();
        System.out.println("Création d'une session mail");
        Session sess = Session.getDefaultInstance(prop, null);
        try
        {
            String user = "vilvens";
            String pwd = "EtPuisQuoiEncore";
            System.out.println("Obtention d'un objet store");
            Store st = sess.getStore("imap"); ←
            st.connect(host, user, pwd);
            System.out.println("Obtention d'un objet folder");
            Folder f = st.getFolder("INBOX");
            f.open(Folder.READ_ONLY);
            System.out.println("Obtention des messages");
            Message msg[] = f.getMessage();
            System.out.println("Nombre de messages : " + f.getMessageCount());
            System.out.println("Nombre de nouveaux messages : " + f.getNewMessageCount());
            System.out.println("Liste des messages : ");
```

Différence par rapport à POP3, juste préciser IMAP pour créer l'objet STORE: recherche dans le fichier: « java.default.providers » la ligne correspond à IMAP.

## 5.4 IMAP: en JAVA (Réception simple mail texte)

```
for (int i=0; i< msg.length; i++)  
{  
    System.out.println("\n< nHeaders du message n°" + (i+1));  
    Enumeration e = msg[i].getAllHeaders();  
    Header h = (Header)e.nextElement();  
  
    while (e.hasMoreElements())  
    {  
        System.out.println(h.getName() + " --> " + h.getValue());  
        h = (Header)e.nextElement();  
    }  
    System.out.println("Texte : " + (String)msg[i].getContent());  
}  
System.out.println("Fin des messages");  
}  
...  
}  
}
```

**Pas de différence avec POP3 sauf !** Lors de la suppression en POP on fait un set flag pour dire qu'on désire supprimer le message et à la fin de la session, il est détruit. En IMAP, il y a une commande **expunge**:

```
public abstract Message[] expunge()throws MessagingException
```