



19/10/2023

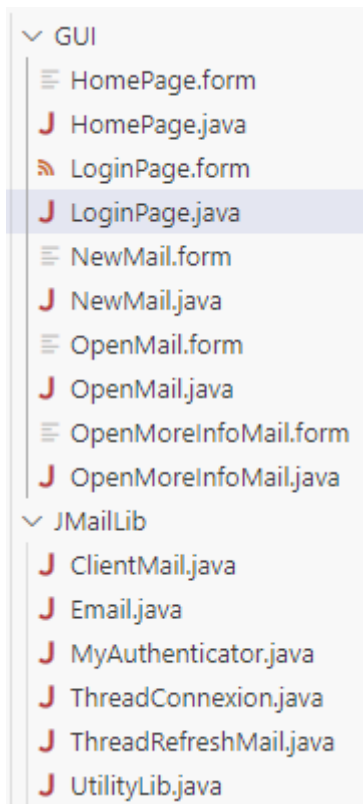
Projet : Java Mail user agent

Lucas Siracusa
Thomas Bastin

Table des matières

Login.....	3
Création de la session.....	4
Création Classe Email.....	6
Récupération de Messages et de Fichiers.....	7
Simple.....	8
Multipart.....	8
Enregistrement d'un fichier attaché.....	10
Envoi d'un mail.....	10
Simple.....	12
Multipart.....	12

Structure du projet :



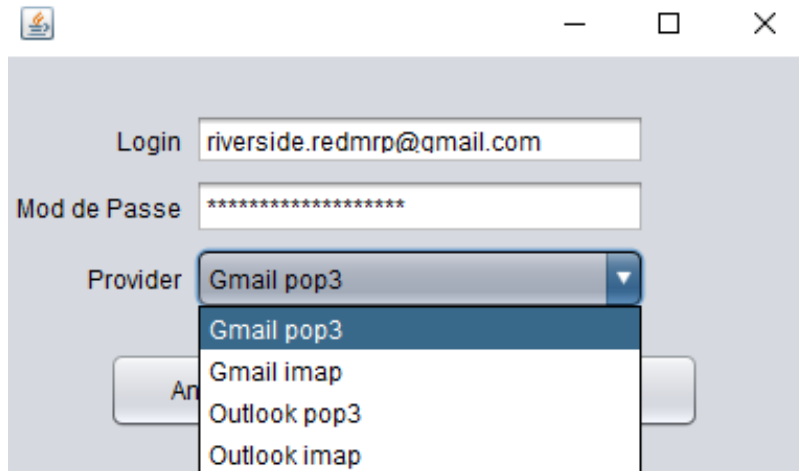
Le projet est décomposé en deux packages, Le premier contiendra toutes les classes du GUI (et les forms).

L'autre package contiendra des classes de logique métier, telles que ClientMail, Email, 2 Threads et une classe utilitaire.

Les pages d'interface elles sont assez basique, une pour le login (qui sera le point d'entrée du projet), une homepage pour afficher les emails, une page pour afficher un mail spécifique et une page pour créer son propre email.

Pour la gestion du projet et de la compilation nous avons utilisé maven qui est un outil moderne permettant de récupérer des dépendances facilement.

Login



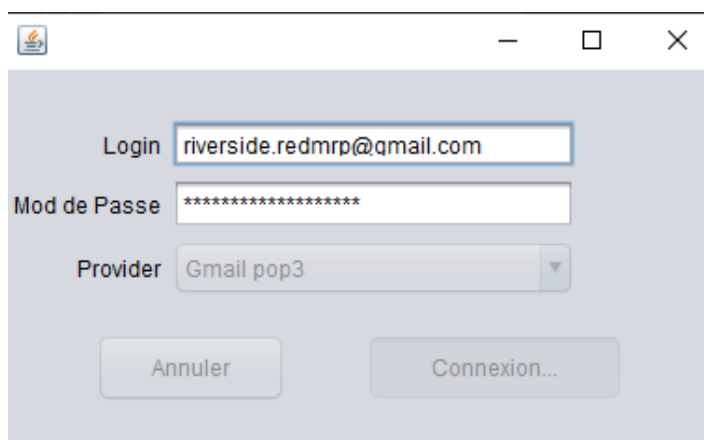
The screenshot shows a Java Swing window titled 'Login'. It contains three text input fields: 'Login' with the value 'riverside.redmrp@gmail.com', 'Mod de Passe' with masked characters '*****', and 'Provider' with 'Gmail pop3' selected. The 'Provider' dropdown menu is open, showing a list of options: 'Gmail pop3' (highlighted), 'Gmail imap', 'Outlook pop3', and 'Outlook imap'. There are two buttons at the bottom: 'Annuler' (disabled) and 'Connexion...' (disabled).

La page de login va permettre de configurer la connexion en choisissant le provider et le type de connexion utilisé pour récupérer les emails (pop3/IMAP). Lors du clic sur le login un thread va se lancer pour exécuter la fonction `doConnexionLogic` qui va instancier l'objet `ClientMail (Session)` ce qui évitera de freezer l'UI.

```
session = new ClientMail(host, prot, login.getText(), Password.getText());
```

Traitement Principal du thread :

```
switch(p[0]) {  
    case "Outlook":  
        host = "smtp-mail.outlook.com";  
        break;  
    case "Gmail":  
        default:  
        host = "smtp.gmail.com";  
}
```



The screenshot shows the same 'Login' dialog box, but now the 'Connexion...' button is enabled and highlighted. The 'Annuler' button remains disabled. The input fields and the 'Provider' dropdown menu are still present and unchanged.

Une fois la connexion établie, le thread va ouvrir la 'HomePage' et fermer la 'LoginPage'.

Création de la session

Nous avons créé un objet session, qui contiendra les différentes informations utiles durant les communications entre le réseau et le client.

```
public class ClientMail {  
    // <editor-fold defaultstate="collapsed" desc="Properties">  
    static String charset = "iso-8859-1";  
  
    private String ident;  
    private Session _session;  
    private Store _store;  
    private Folder _folder;
```

Il contiendra l'objet session de JavaMail permettant de se connecter sur du pop3 ou de l'IMAP et faire du SMTP. On l'utilisera pour accéder au Store et se connecter sur ce dernier, afin de récupérer l'objet Folder. Ce dernier contient une méthode pour récupérer la liste des messages de votre boîte en PoP3 ou IMAP.

Le constructeur de la classe ClientMail va instancier les différents composants et choisir les bons paramètres de connexion en fonction du type de connexions choisi au login.

```
//Creation d'un object Authenticator (classe anonyme)  
Authenticator conn = new MyAuthenticator(ident, password);  
  
this.ident = ident;  
  
//Creation d'un objet session basé sur les props et l'authenticator.  
System.out.println("Session Created");  
_session = Session.getInstance(props, conn);  
  
//Recuperation du store  
System.out.println("Store Created");  
_store = _session.getStore(protocol);  
  
//Connexion au store  
_store.connect(serveurReception, Integer.decode(port), ident, password);  
System.out.println("Connect on: " + serverHost + ", Port: " + port + ", :");  
  
//Recuperation du folder INBOX et ouverture de ce dernier.  
_folder = _store.getFolder("INBOX");  
_folder.open(Folder.READ_ONLY);
```

On remarque que le folder a été ouvert en readonly pour éviter toutes erreurs de suppression.

ClientMail embarque plusieurs fonctions :

```
public ArrayList<Email> GetListMail() throws MessagingException, IOException{  
  
    public int GetMessageCount() throws MessagingException{  
  
    public void Close() throws MessagingException{
```

GetListMail permet de récupérer une liste d'objets Email, provenant du Store, qui contiendra toutes les informations nécessaires au bon traitement des Email reçus.

```
        Message[] msg = null;  
        ArrayList<Email> list = new ArrayList<>();  
  
        _folder.close(true);  
        _folder = _store.getFolder("INBOX");  
        _folder.open(Folder.READ_ONLY);  
  
        msg = _folder.getMessages();  
  
        //Loop on array to init new Email();  
        for(Message m : msg){  
            //Add for each elements a new Email based on message  
            Email tmp = new Email((MimeMessage) m);  
            list.add(tmp);  
        }  
        Collections.reverse(list);  
  
        return list;
```

GetMessageCount permet de récupérer le nombre de message actuellement dans l'Inbox

Le Close permet de fermer les différentes connexions de manière correcte.

Création Classe Email

Pour récupérer des messages contenus dans l'Inbox, nous avons créé une classe Email qui va contenir des méthodes et variables simplifiant la manipulation des Email reçus pour le développeur. Ils encapsulent MimeMessage.

```
private String _id;
private String _from;
private ArrayList<String> _to;
private ArrayList<String> _CC;
private String _subject;
private String _message;
private ArrayList<Header> _headers;
private ArrayList<String> _filePaths;

private MimeMessage source;
```

Cette classe contiendra diverses fonctions et variables (setters, getters, constructeurs, ...) afin de pouvoir gérer la sauvegarde d'un fichier ou encore la création complète de la chaîne d'un caractère d'un mail (l'envoyeur, destinataire, headers, sujet, titre, contenu, fichier(s)).

Le constructeur (utilisé dans la méthode GetListMail de la classe ClientMail), prend en paramètre un MimeMessage et décompose les éléments qui nous intéressent afin de créer un objet Email. Il utilise des setters particuliers qui permettent de convertir. Les données de la librairie JavaMail, en données plus standard et facilement manipulables pour le développeur.

```
public Email(MimeMessage message) throws MessagingException, IOException {
    setId(message.getMessageID());
    setFrom(message.getFrom());
    setTo(message.getRecipients(RecipientType.TO));
    setCC(message.getRecipients(RecipientType.CC));
    setSubject(message.getSubject());
    setMessage(message);
    setFilePaths(message);
    setHeaders(message.getAllHeaders());

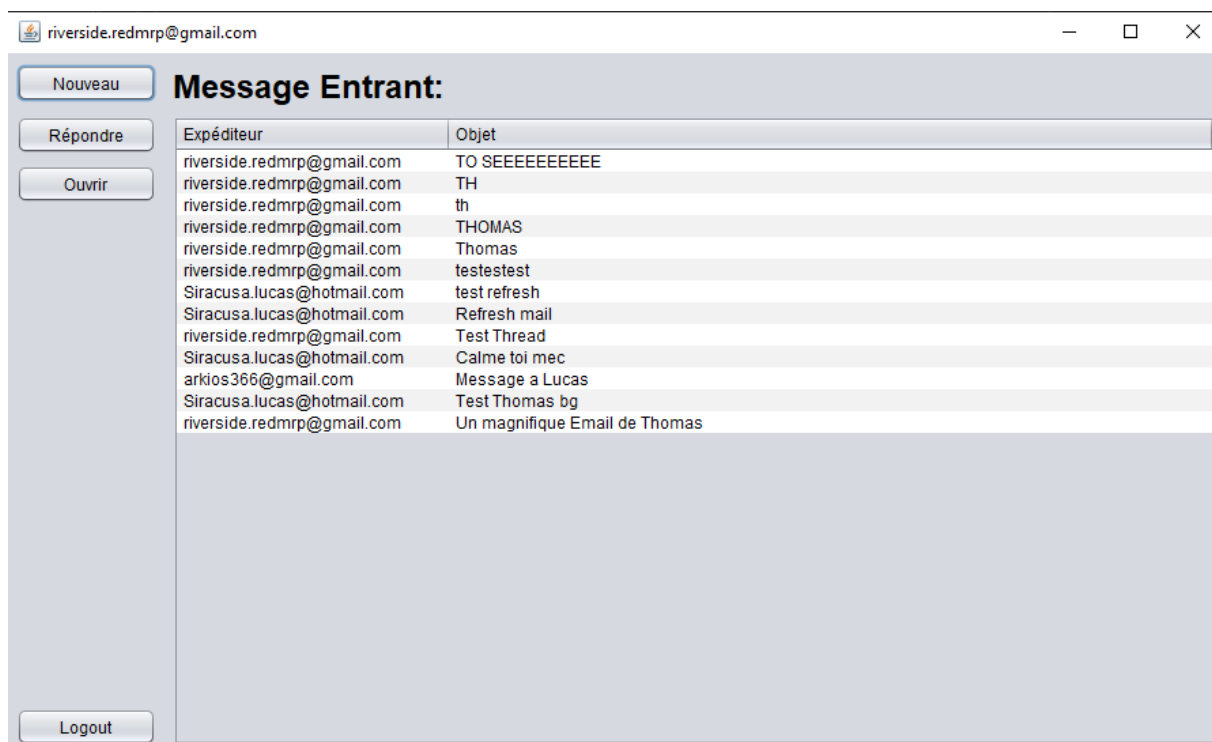
    source = message;
}
```

Par exemple : Une fonction qui convertit l'array des adresses d'envoi en une ArrayList de String

```
private void setTo(Address[] _to) {
    ArrayList<String> tmp = new ArrayList<>();
    if(_to == null){
        this._to = tmp;
        return;
    }

    for(Address row : _to){
        tmp.add(((InternetAddress)row).getAddress());
    }
    this._to = tmp;
}
```

Récupération de Messages et de Fichiers



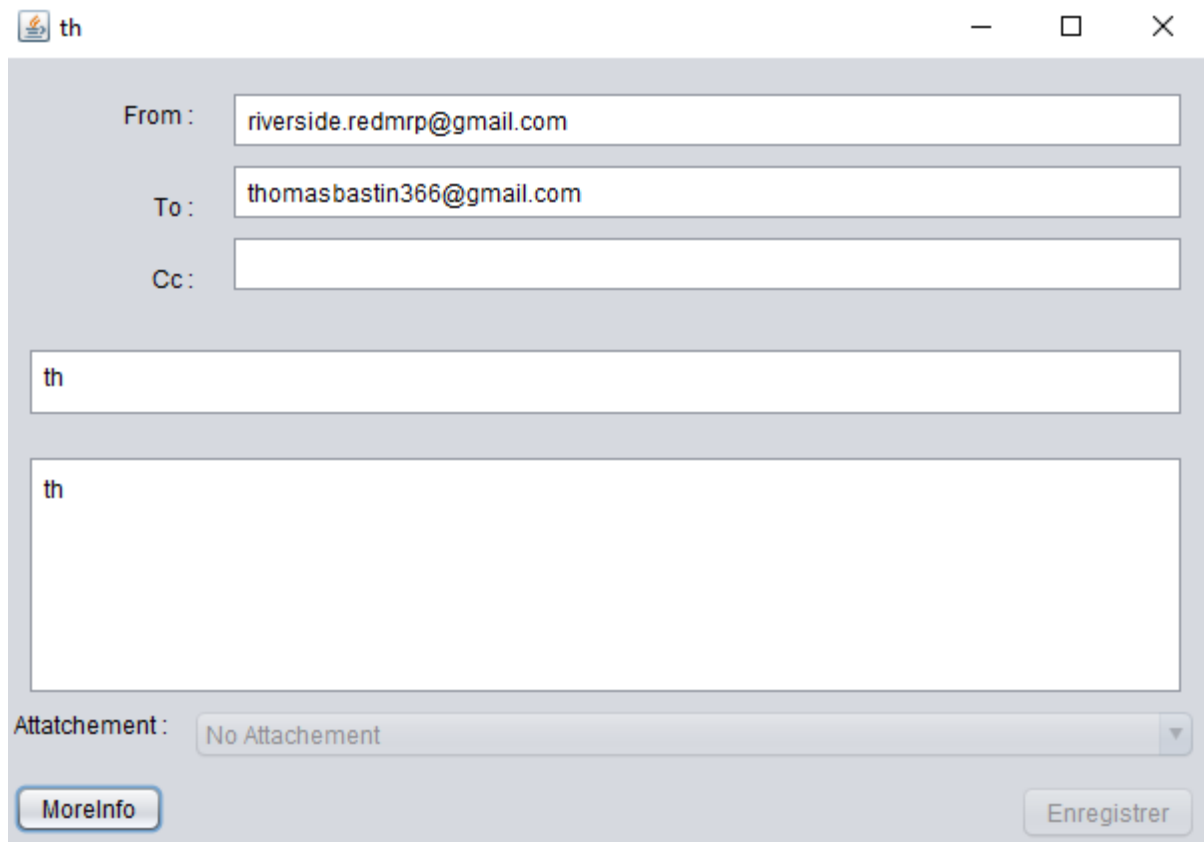
Pour récupérer un message provenant d'un MimeMessage nous utilisons la méthode setMessage() de la classe Email, lors de la construction de l'objet. Cette fonction va être chargée de récupérer le texte de l'email, qu'il soit SimplePart ou MultiPart.

```
private void setMessage(MimeMessage _message) throws MessagingException, IOException {
```


Simple

Dans le cas où le Message n'est pas multipart, on va simplement décoder le texte qui sera récupéré via la méthode `getContent()` de la classe `MimeMessage`.

```
//RECUPERATION MESSAGE SIMPLEPART
if(!_message.isMimeType("multipart/*")){
    ReturnedMessage = MimeUtility.decodeText((String) _message.getContent());
    this._message = ReturnedMessage;
    return;
}
```



th

From : riverside.redmrp@gmail.com

To : thomasbastin366@gmail.com

Cc :

th

th

Attachement : No Attachment

MoreInfo Enregistrer

Multipart

Dans le cas d'un multipart, on utilise une boucle pour lire tous les Parts de l'email. Si un Part est type « text/plain » (Et que ce n'est pas un fichier). On lit son contenu. Dans le cas où ne trouve pas de Part correspondant à cela, on doit alors chercher un Part de type « MultiPart/Alternative ». Une fois trouvé, on doit récupérer son contenu dans une variable Multipart, et effectuer à nouveau l'opération pour trouver un « text / plain ». Ce Part « text/plain » dans les deux cas sera casté en string et utilisé comme message.

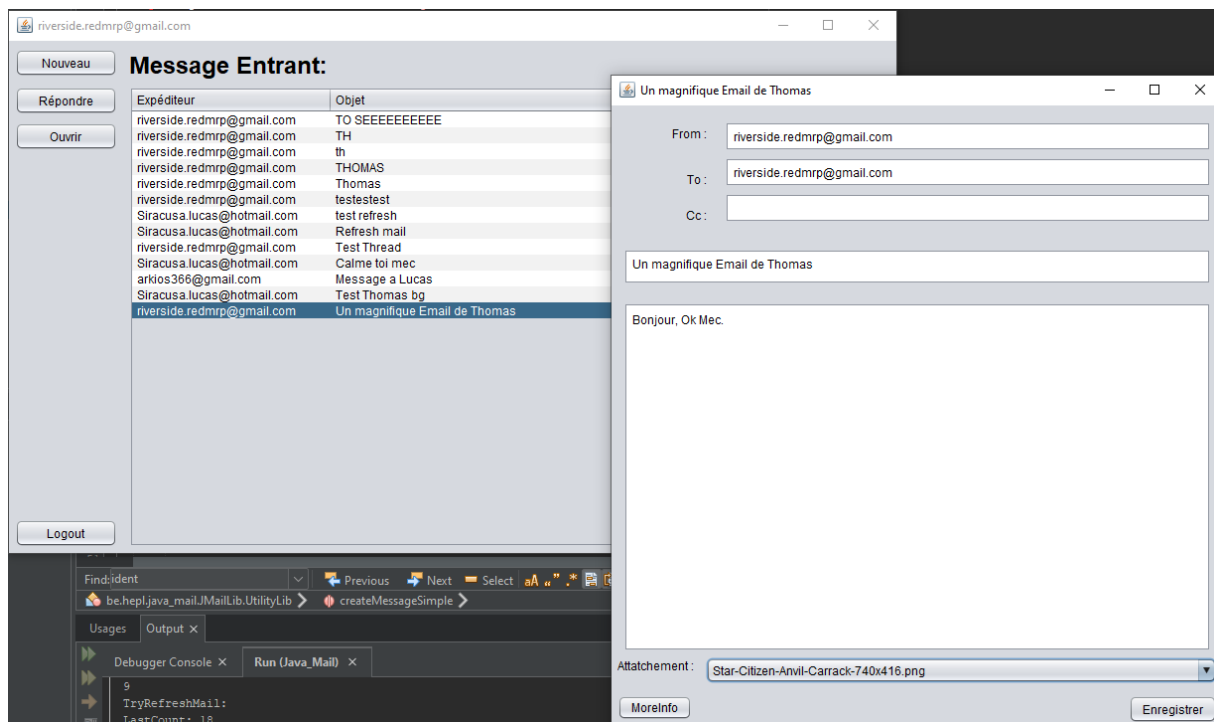
```

//RECUPERATION MESSAGE MULTIPART
for(int i = 0 ; i<n ; i++){
    Part part = MultiMsg.getBodyPart(i);          //get a nested part

    //Message Principal en Directe
    if(part.isMimeType("text/plain") && part.getFileName() == null) {
        ReturnedMessage = MimeUtility.decodeText((String) part.getContent());
        break;
    }
    //Message Principal encapsulé dans un autre multipart (outlook)
    else{
        if(part.isMimeType("multipart/alternative")){
            //Recuperation du nested MultiPart
            Multipart NestedMP = (Multipart) part.getContent();
            int n2 = NestedMP.getCount();          //Recuperation nombre de part

            //On loop dans le multipart a la recherche d'un textplain
            for(int j = 0 ; j<n2 ; j++){
                Part part2 = NestedMP.getBodyPart(0);
                if(part2.isMimeType("text/plain") && part2.getFileName() == null){
                    ReturnedMessage = MimeUtility.decodeText((String) part2.getContent());
                    break;
                }
            }
        }
    }
}
}

```



Il est également requis de vérifier s'il existe ou non des fichiers pouvant être téléchargés par l'utilisateur. Pour ce faire nous allons utiliser une autre méthode de la classe Email.

```
private void setFilePaths(MimeMessage Message) throws MessagingException,
```

On va à nouveau itérer sur les parts du mail, mais ce coup si en cherchant tout part qui est un fichier. Une fois trouvé on ajoute à notre liste de fichier le nom du fichier afin de l'utiliser plus tard pour télécharger le fichier en local.

```
//Check if it's a file
if (part.getDisposition() != null && part.getDisposition().equalsIgnoreCase(Part.ATTACHMENT)) {
    String FileName = MimeUtility.decodeText(part.getFileName());

    ReturnedFilePaths.add(FileName);
}
```

```
//Check if it's a file
if (part.getDisposition() != null && part.getDisposition().equalsIgnoreCase(Part.ATTACHMENT)) {
    String FileName = MimeUtility.decodeText(part.getFileName());

    ReturnedFilePaths.add(FileName);
}
```

Enregistrement d'un fichier attaché

Pour enregistrer le fichier sur le disque, l'utilisateur sélectionne lors de la lecture du mail avec l'interface utilisateur un path où enregistrer un des fichiers de l'Email.

On va alors utiliser une petite méthode pour effectivement enregistrer le fichier en local. La méthode est embarquée dans la classe Email pour plus de simplicité.

Elle va rechercher dans les Mimeparts le part contenant le fichier à télécharger, puis place le fichier en mémoire via un buffer pour finalement l'enregistrer sur disque.

```
//Si le part est un fichier et que le nom correspond, on récupère une référence vers son flux d'entrée
if(MimeUtility.decodeText(part.getFileName()).equals(fileName)){

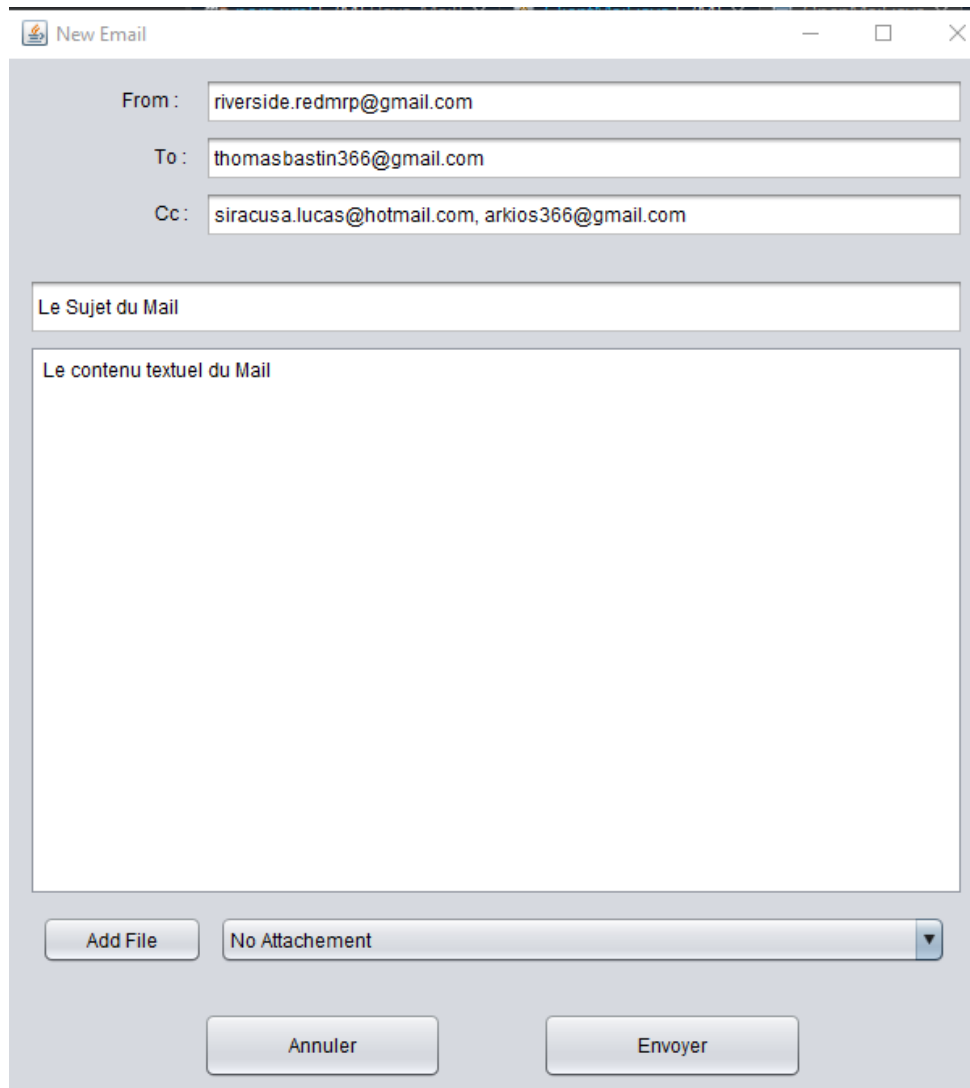
    System.out.println("Fichier à télécharger : " + MimeUtility.decodeText(part.getFileName()));

    //Connexion avec le serveur pour récupérer le fichier
    InputStream readStream = part.getInputStream();
    //Endroit où il va falloir enregistrer le fichier
    FileOutputStream writeStream = new FileOutputStream(directory);
    //Temporary Stream
    ByteArrayOutputStream tmp = new ByteArrayOutputStream();

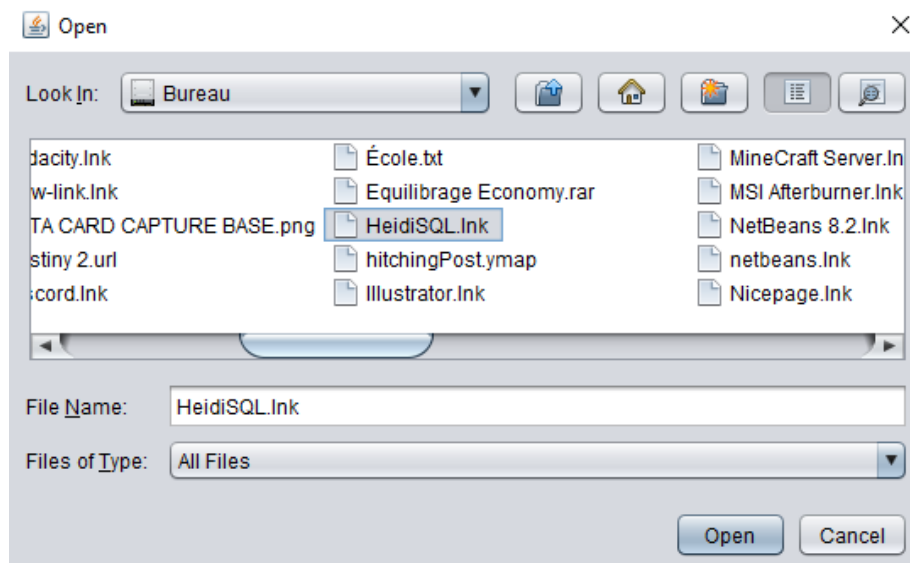
    int c;
    //On met l'inputStream dans le buffer temporaire
    while ((c = readStream.read()) != -1) {
        writeStream.write(c);
    }

    //On écrit depuis le stream tampons dans le fichier
    tmp.writeTo(writeStream);

    //On ferme tout les flux
    writeStream.close();
    tmp.close();
    readStream.close();
}
```



Si on le souhaite on peut ajouter des fichiers attachés à notre Email en cliquant sur Add File.



Simple

Si aucun fichier n'est attaché à notre message nous l'enverrons en SimplePart en utilisant la fonction utilitaire createMessageSimple().

```
public static void createMessageSimple(MimeMessage mail, Address[] To, Address[] Cc, String Subject,
    System.out.println("Création Message Simple");
    mail.setFrom();

    //Put the To List into the MimeMessage Object
    mail.setRecipients(Message.RecipientType.TO, To);

    //Put the Cc List into the MimeMessage Object
    mail.setRecipients(Message.RecipientType.CC, Cc);

    //Define Object
    mail.setSubject(Subject);

    //Define MainMessage
    mail.setText(MimeUtility.encodeText(Text));
    mail.setContent(null);
}
```

Multipart

Si des fichiers ont été attachés à notre Email, alors on enverra un fichier MultiPart. Pour le savoir on regarde si la liste de fichier de l'interface utilisateur contient autre chose que le "No Attachment".

À chaque fois que l'on clique sur addFile, on ajoute un fichier à la liste des fichiers à envoyer. En parallèle on ajoute un Part contenant le fichier dans le MultiPart (qui lui-même est dans le MimeMessage).

```
/*Ajoute les fichier joints au message principal*/
public static void setFilePart(Multipart Multip, String FilePath, ArrayList<String>
    String[] Names = FilePath.split("\\\\");
    String Name = Names[Names.length - 1];

    MimeBodyPart BodyPart = new MimeBodyPart();

    //Idiqué le type de multipart
    BodyPart.attachFile(new File(FilePath));
    BodyPart.setDataHandler (new DataHandler(new FileDataSource (FilePath)));
    BodyPart.setFileName(Name);

    Multip.addBodyPart(BodyPart);

    FileList.add(Name);
}
```

Une fois que l'on clique sur le bouton envoyé, une dernière fonction utilitaire est appelée afin de rajouter le contenu textuel, et les autres données dans le MimeMessage.

```
public static void createMessageMultiPart(MimeMessage mail, MultiPart multipart) {

    mail.setFrom();

    mail.setRecipients(Message.RecipientType.TO, To);
    mail.setRecipients(Message.RecipientType.CC, Cc);

    mail.setSubject(Subject);

    System.out.println("Création Message MultiPart");
    //Ajout du Texte comme premier composant
    MimeBodyPart msgBP = new MimeBodyPart();
    msgBP.setText(MimeUtility.encodeText(Text));
    multipart.addBodyPart(msgBP);

    //mail.setContent(multipart);
}
```

Une fois le MimeMessage finalisé et prêt à l'envoi, nous l'envoyons en utilisant le réseau SMTPs au serveur de Mail du client connecté.

```
try {
    Transport transport = session.getSession().getTransport("smtps");
    transport.send(mail);
    transport.close();
} catch (MessagingException ex) {
```