# Projet de Java Licence - JButcher

Le but de ce projet est de réaliser un filtreur de mail. Le logiciel devra tourner au moins sur les environnements Unix, MacOS X et Windows indifféremment.
forax@univ-mlv.fr, pierre.peterlongo@univ-mlv.fr, benoit.olivieri@univ-mlv.fr

## Description du projet

Le programme `JButcher` est un filtreur de mail. L'architecture de cet outils devra être assez générique pour permettre à n'importe qui de "customiser" le filtreur pour effectuer différents traitements sur des mails.
Il faudra que vous gardier à l'esprit que `JButcher` devrait permettre de :

- rediriger ou relayer les mail un peu comme la fait le logiciel `procmail`.
- de trouver les spams et de l'indiquer comme tel pour un lecteur de mail comme le ferait un logiciel anti-spam comme par exemple `SpamAssassin`.
- laisser la possibilité à n'importe qui d'ajouter de nouvelles fonctionnalités sous forme de filtre ou d'action.

## Modèle et Concept

`JButcher` manipule trois concepts, les règles, les filtres et les actions.
Une règle définie :

- une source de mails
- une filtre à appliquer pour chaque mail
- un seuil (une valeur à virgule flottante)
- un ensemble d'actions

Les mails peuvent être obtenus suivant les protocoles POP et IMAP à partir de la source de mails. L'ensemble des actions est effectuées si le filtre (dit filtre de départ ou `startFilter`) est positif, c'est à dire si le filtre dépasse une valeur seuil fixée dans la règle pour chaque mail de la source de mails.
Un filtre de mail analyse un mail en provenance de la source de mails et renvoie une valeur réelle.
Il est possible, de plus, de spécifier des relation entre des filtres et de construire avec cela un arbre de filtre. Par exemple, le filtre `ORFilter` sera en relation avec d'autres filtres pour effectuer un ou logique entre les resultats de ceux-ci.
L'ensemble des actions à effectuer si pour un mail le résultat du filtre `startFilter` est supérieur au seuil. `JButcher` déclenche alors l'ensemble des actions qui s'effectuent sur ce mail.
`JButcher` est initialisé par l'intermédiaire d'un fichier de configuration (`butcher.conf`) définissant un ensemble des règles avec leurs filtres et leurs actions.
La section "Exemples de fichier de configuration" détaille deux exemples de configurations.

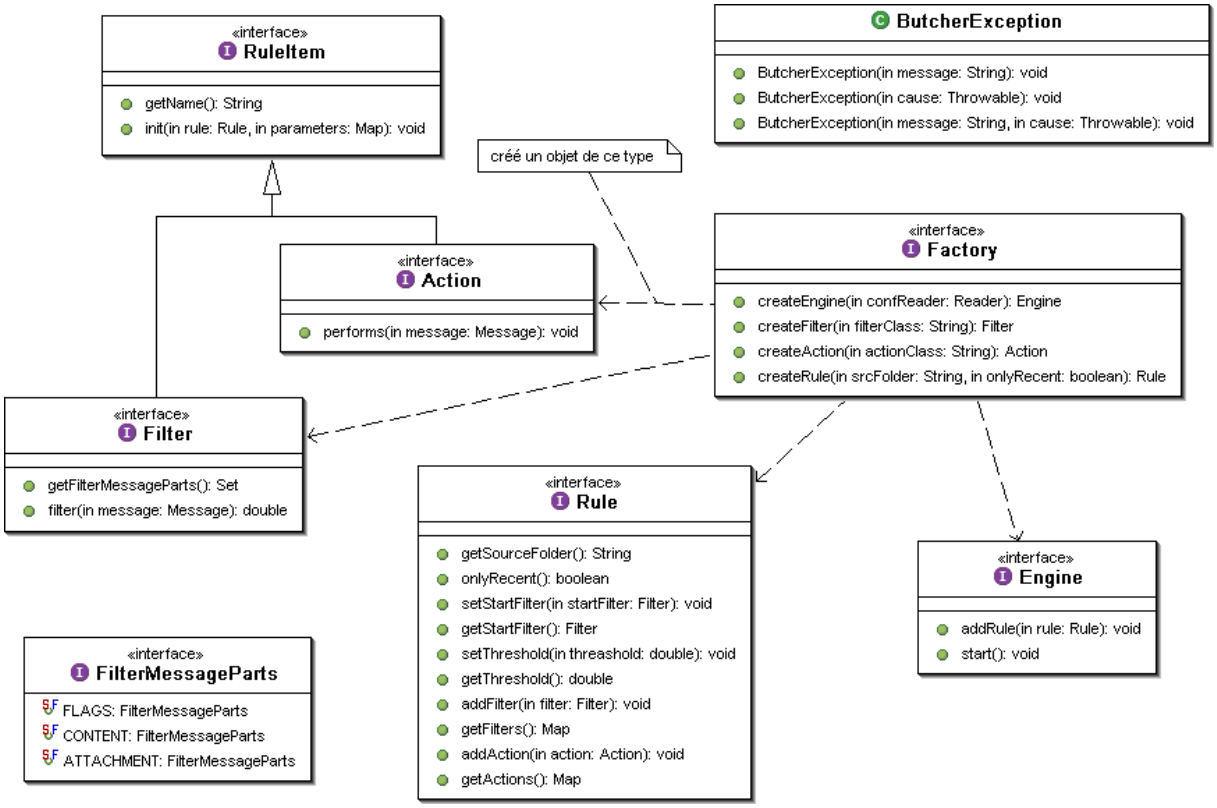
## Architecture proposée

Lors du premier rendu, vous avez chacun proposé une architecture pour `JButcher`.
Pour le second rendu, c'est à dire le développement de l'application `JButcher`, nous vous

demandons d'écrire un ensemble de classes implantant des interfacee prédéfinies.
Cette section à pour but de présenter ces interfaces :

- L'interface `Factory` contient les méthodes permettant de créer tous les objets de `JButcher`. Nous utilisons ici une interface pour permettre à chacun de vous de fournir sa propre implémentation.
- L'interface `Engine` correspond au moteur de JButcher, c-a-d la partie de code qui va aller chercher les mails et appliquer les filtres dessus.
- L'interface `Rule` correspond à une règle du fichier de configuration. Une règle contiendra donc les filtres et les actions.
- L'interface `RuleItem` représente la partie commune entre les objets `Action` et `Filter`. Cela permet d'écrire du code manipulant l'un ou l'autre indifféremment.
- L'interface `Action` correspond à une action.
- L'interface `Filter` correspond à un filtre.
- L'interface `FilterMessageParts` permet d'indiquer une partie ou plusieurs de mail. Cette interface est en relation avec la méthode `getFilterMessageParts()` de l'interface `Filter`.
  Ce mécanisme est optionel dans l'implantation qui vous est demandé. Il permet à un filtre d'indiquer quelles sont les parties de mail qui l'intéresse. Cela permet à l'`Engine` d'effectuer des optimisations sur les parties de mail qu'il doit fournir au filtre, et donc de demander à un fournisseur de mails, que les parties de mail qui interrese le filtre.

Voici le lien sur le fichier ZIP contenant les sources des interfaces : jbutcher.zip (`jbutcher.zip`) .
Le graphique ci-dessous, est un diagramme de classe UML indiquant les dépendances et relations entre les interfaces présentées ci-dessus.




## Exemples de fichier de configuration

```
   <butcher-conf>
        <rules>
          <rule src="imap://but.cher:butcher@imap.laposte.net/INBOX"
               onlyRecent="false"
               threshold="20.0"
               start-filter="sum">
          <filters>
           <filter name="sum"
                      class="fr.umlv.jbutcher.example.SumFilter"
                      list="test,test2"/>
           <filter name="test"
                  class="fr.umlv.jbutcher.example.TestFilter"
                  threshold="5.0"/>
           <filter name="test2"
                  class="fr.umlv.jbutcher.example.TestFilter"
                  threshold="15.0"/>
            </filters>
           <actions>
            <action name="print"
class="fr.umlv.jbutcher.example.PrintAction"/>
            <action name="print2"
class="fr.umlv.jbutcher.example.PrintAction"/>
           </actions>
          </rule>
         </rules>
   </butcher-conf>
```

Ici, on récupère par IMAP tous le mails et ceux-ci sont affichés deux fois sur la sortie standard.

```
   <butcher-conf>
          <rules>
           <rule src="pop3://but.cher:butcher@pop.laposte.net/INBOX"
                 onlyRecent="true"
                 threshold="10.0"
                 start-filter="anti-spam">
            <filters>
             <filter name="anti-spam"
                     class="fr.umlv.jbutcher.example.AntiSpamFilter"
                     file="bad-words.txt"/>
            </filters>
            <actions>
             <action name="forward"
class="fr.umlv.jbutcher.example.ForwardAction"
                     email-address="forax@univ-mlv.fr"
                     smtp-host="smtp.laposte.net"
                     smtp-port="25"/>
            </actions>
           </rule>
          </rules>
   </butcher-conf>
```

Ici, on récupère par POP les mails récents et ceux-ci sont transférés (forward) à l'adresse mail "forax@univ-mlv.fr" par le serveur SMTP "smtp.laposte.net" si ceux-ci sont considérés comme du spam.
Exemple de fichier "bad-words.txt"

```
   viagra(5.2)
   xanax(5.0)
   remi(-2.2)
```

Les valeurs mis entre parenthèse indique les valeurs attribuées à chaque mot.

## Liste des filtres et actions demandées

Liste des filtres à implanter :

1. `TestFilter` Paramètre : `threshold` : un seuil.
   Renvoie toujours la valeur du seuil passée en paramètre quelque soit le mail.
2. `OrFilter`
   Paramètre : `threshold` : un seuil.
   Paramètre : `list` : une liste de nom de filtre.
   Filtre dont la valeur est égale au seuil si un des filtres de la liste a une valeur supérieure au seuil. Sinon la valeur du filtre est zéro.
3. `AndFilter`
   Paramètre : `threshold` : un seuil.
   Paramètre : `list` : une liste de nom de filtre.
   Filtre dont la valeur est égale au seuil si tous des filtres de la liste ont une valeur supérieure au seuil. Sinon la valeur du filtre est zéro.
4. `SumFilter`
   Paramètre : `list` : une liste de nom de filtre.
   Filtre dont la valeur est égale à la somme des filtres de la liste.
5. `SpamFilter`
   Paramètre : `file` : fichier contenant un mot et sa valeur par ligne.
   Filtre dont la valeur est égale à la somme des valeurs des mots reconnus dans le fichier.

Liste des actions à implanter :

1. `PrintAction`
   Affiche le contenu du mail filtré.
2. `ForwardAction`
   Paramètre : `email-address` adresse mail
   Paramètre : `smtp-host` machine serveur SMTP
   Paramètre : `smtp-port` port SMTP
   Renvoie un mail filtré vers une adresse mail.
3. `MarkAction`
   Paramètre : `flag` nom du drapeau
   Paramètre : `flag-value` valeur du drapeau
   Ajoute à un mail filtré un drapeau (flag) dans son entête, pour indiquer que celui-ci est un spam.

## Calendrier

Ce projet est à faire par binôme (cela veut dire deux personnes pas trois ni une). Le rendu du projet est découpé en deux parties :

1. Pré-rapport indiquant l'architecture objet et tests d'utililisation.
   Date de rendu : le 4 avril, avant minuit.
2. Rendu du projet en lui-même.
   Date de rendu : le 21 mai, avant minuit.

Le rendu s'effectuera par mail sous forme d'une pièce jointe au format ZIP contenant l'ensemble des programmes et documents. Le mail devra être envoyé avant minuit aux trois adresses données au début de ce document.

## Détail du second rendu

Le second rendu correspond à un programme ecrit en java livré sous forme d'un jar exécutable ainsi que deux documentations (utilisateurs et développeurs).
Voici le nom des repertoires et fichiers qui doivent être contenus dans l'archive ZIP.

1. un fichier `readme.txt` indiquant comment compiler et exécuter le programme, où se trouve la doc, etc.
2. un fichier ANT `build.xml` permettant de compiler les sources du programme jbutcher et de créer le jar exécutable.
3. un répertoire `src` contenant l'ensemble des sources (.java) sous forme de plusieurs paquetages.
4. un répertoire `classes` contenant l'ensemble des classes (.class) correspondant aux sources sous forme de plusieurs paquetages.
5. un répertoire `lib` contenant l'ensemble des jars (entre autres ceux de `JavaMail`) plus le jar exécutable correspondant à l'application `jbutcher.jar`.
6. un répertoire `bin` contenant deux fichiers, `jbutcher.sh` démarrant le logiciel sous linux (en fait unix) et `jbutcher.bat` demarrant le logiciel sous Windows (attention à NT).
7. un répertoire `conf` contenant en plus des deux fichiers de configuration ci-dessus, d'autres exemples de configuration.
8. un répertoire `docs` contenant deux documents au format PDF :
    1. La documentation utilisateur `user.pdf` contenant en plus des informations classiques (comment compiler, exécuter, etc.) une description de l'ensemble des actions et filtres implantés ainsi que les paramètres de ceux-ci et des exemples de fichiers de configuration (autres que ceux proposés dans ce document).
    2. La documentation développeur `dev.pdf` indiquant l'architecture complète de votre JButcher ainsi qu'un `How-to` permettant de créer facilement soi-même sa propre action ou son propre filtre.

    En plus des deux documents, le répertoire `docs` devra contenir un sous-répertoire `api` contenant la documentation complète du logiciel au format javadoc. Vous pouvez vous aider des sources des interfaces pour voir comment il faut commenter vos propres sources.

## Références


- JavaMail (`http://java.sun.com/products/javamail/`)
- Java API for XML Processing (JAXP) (`http://java.sun.com/xml/jaxp/index.jsp`)
- Expression régulière (`http://java.sun.com/j2se/1.4.2/docs/api/java/util/regex/package-summary.html`)
- Jar exécutable (`http://java.sun.com/j2se/1.4.2/docs/guide/jar/jar.html`)
- Ant (`http://ant.apache.org/`)