

todo2email

Projet 1 de Java ¹

Remarques :

- ↪ Date de remise de l'énoncé du projet : semaine du 13 février 2006
 - ↪ Date **buttoire** de remise du rapport : semaine du 6 mars 2006
- Renseignez-vous auprès de votre professeur afin de connaître ses modalités de remise avant la date fatidique².
- N'oubliez pas non plus de consulter le document "Méthode d'évaluation" (evaluation.pdf).

COMME tout un chacun, vous avez quantité de choses à faire et à ne pas oublier. Ne serait-il pas pratique que chaque matin, voire chaque heure un (ou plusieurs) mail(s) vienne(nt) vous rappeler telle ou telle chose ?

C'est ce que nous nous proposons de faire dans ce projet. Nous allons implémenter un programme permettant d'envoyer des *todos* par mail. Le programme prendra en paramètre une action à exécuter parmi *run*, *new*, *add*, ..., sans paramètre, il nous affichera une aide sommaire du style ;

```
Parametres insuffisants

todo2email: Gere tes TODOs et te les emaile

Usage: java Todo2email action [option]
Actions:
    new tonemail (cree un nouveau compte)
    email tonnouvelemail (met à jour ton email)
    run
    add tododescription todoecheance
    list
```

- ↪ *new*, crée un nouveau compte et détermine l'adresse email à laquelle envoyer les *todos*,
- ↪ *email*, permet de changer l'adresse email à laquelle sont envoyés les *todos*,
- ↪ *run*, parcourt la liste des *todos* et envoie par mail ceux qui arrivent à échéance³,
- ↪ *add*, ajoute un nouveau *todo* à la liste des *todos*,
- ↪ *list*, liste l'ensemble des *todos* sur la sortie standard

¹Inspiré de rss2email de Aaron SWARTZ

²N'oubliez pas qu'une date se compose, dans notre cas, du jour et d'une *heure*

³Pour que le programme soit utile, on placera cet appel dans un tâche *cron* (voir la page de manuel de *crontab*) ... pour ceux qui veulent et qui ont le temps.

1 Les fonctionnalités

1.1 Option *new*

Création d'un nouveau compte. Pour ce faire, il suffit de créer deux fichiers. L'un contenant l'adresse email à utiliser, l'autre, vide, destiné à contenir les données (les *todos*).

1.2 Option *email*

Permet de changer l'adresse email ... il suffira de modifier le fichier contenant ladite adresse.

1.3 option *run*

C'est sans doute là le gros du travail ... il faut

- ↪ lire le fichier de données,
(Cette lecture servira à remplir un objet "collection" contenant les *todos*. Les *todos* sont des objets ayant deux attributs ; *description* et *dueDate* (échéance). La collection peut être une `ArrayList` ou un vecteur (tableau à une dimension) de `ToDoItem` (voir plus loin.)
- ↪ parcourir la collection, pour chaque *todo* de la collection
 - ˘ vérifier si l'échéance est atteinte,
 - ˘ si oui, envoyer le *todo* par mail, et le supprimer de la collection,
 - ˘ si non, ne rien faire
- ↪ réécrire le fichier de données sur base de la collection modifiée

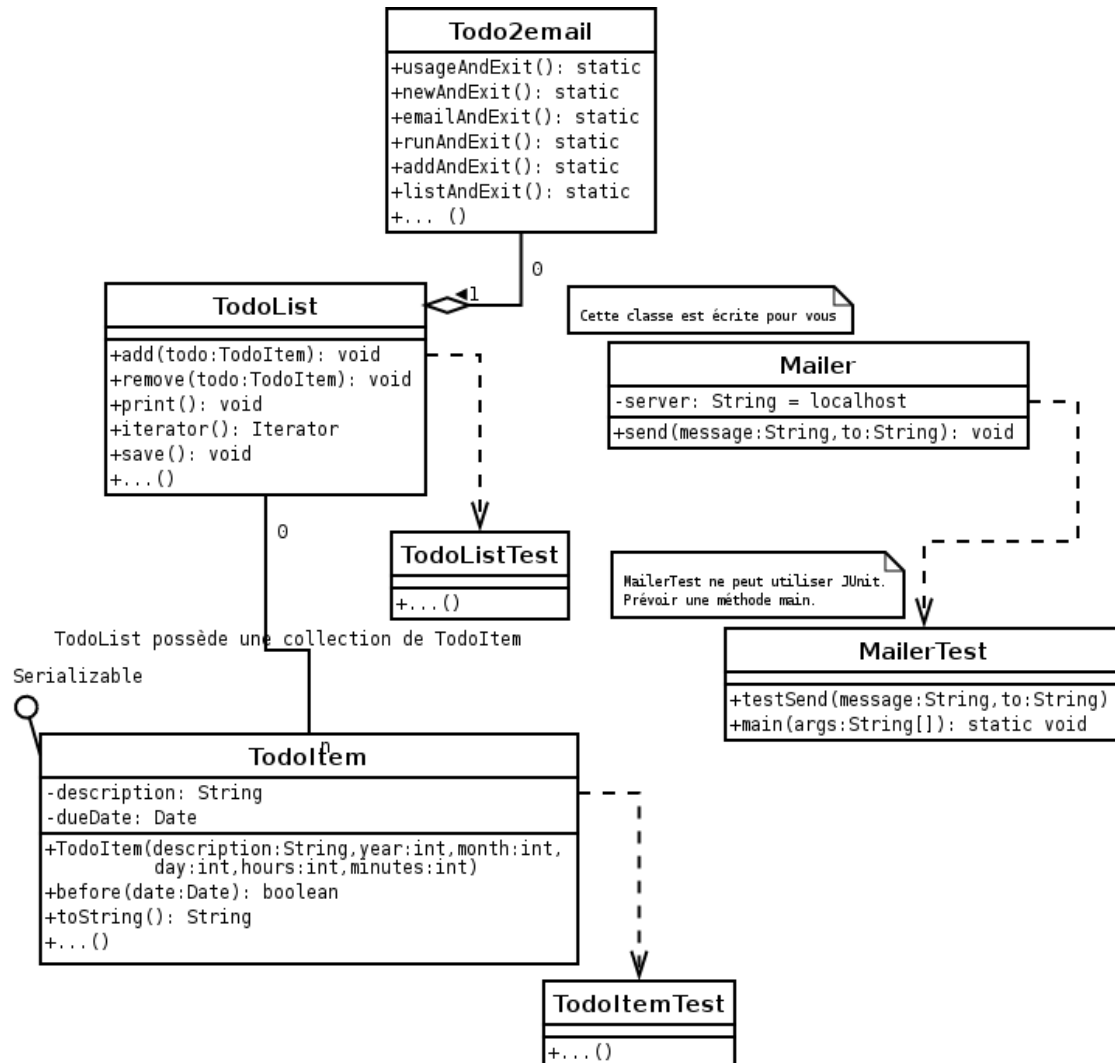
1.4 option *add*

Permet l'ajout d'un *todo*. A nouveau, il faut lire le fichier de données au début et il le sauvegarder à la fin. Entre deux, il suffit d'ajouter le *todo* à la collection (la collection de *todos* ne doit pas être triée).

1.5 option *list*

Permet de lister tous les *todos* à l'écran. Pour ce faire, lire les données, parcourir la collection et afficher chaque *todo* à l'écran. Dans ce cas, il sera inutile de réécrire la collection dans le fichier à la fin.

2 Implémentation



2.1 Implémentation de TodoItem

Cette classe contient deux **attributs** privés, description de type String et dueDate de type Date.

Les TodoItem devront être sauvegardés sur disque (dans un fichier) la classe **doit** donc implémenter *Serializable*. Nous verrons les détails plus tard au cours ... à ce stade, il vous suffit d'écrire

```
public class TodoItem implements Serializable { ... }
```

Pour implémenter le constructeur, vous devrez utiliser la classe *GregorianCalendar*. Voir les méthodes *set*, permettant de déterminer un temps en fonction du jour, du mois, ... et *getTime* qui permet d'obtenir l'objet *Date* correspondant à la date du calendrier.

Vous devrez également **lire** le fichier contenant votre *TodoList* (voir plus bas).

Pour la méthode *before*, il suffit d'utiliser la méthode *before* de la classe *Date* dans la classe *TodoItem*.

La méthode `toString` appelle peu de commentaires. Elle se retrouve dans presque chaque classe et, comme son nom l'indique, elle permet de fournir une représentation textuelle de l'objet. Dans ce cas précis, nous utiliserons cette représentation textuelle dans le corps du mail.

2.2 Implémentation de `TodoList`

Cette classe ne contient qu'un seul attribut, c'est la *Collection* de todos. Nous vous conseillons l'emploi de l'objet `ArrayList` pour implémenter cette collection (vous pouvez utiliser un tableau de todo `TodoItem[]` mais ... non, définitivement nous ne vous le conseillons pas). Les méthodes `add`, `remove`, `iterator` seront très simplement implémentées en utilisant les méthodes correspondantes de la classe `ArrayList`.

La méthode `print` parcourt simplement la collection et affiche à l'écran (via un simple `println` puisque l'on a écrit une méthode `toString`) la liste des *todos*.

La méthode `save` permet de sauver la `TodoList` (voir plus bas).

2.3 Implémentation de `Todo2email`

L'implémentation des méthodes `actionAndExit()` est laissée à votre discrétion. Ces méthodes utilisent majoritairement les méthodes définies dans la classe `TodoList` détaillée ci-dessus.

2.4 Implémentation des tests

Vous mettrez en place une série de tests comme présentés dans le td "**JUnit**". Le test de la classe `Mailer` est quant-à lui particulier en ce sens qu'il faut vérifier l'envoi (et la réception) d'un mail. Cette classe contiendra donc une méthode `main` permettant d'effectuer ces tests.

3 Aspects techniques

3.1 Lire et écrire dans un fichier

Ces notions seront vues en détail au cours ... plus tard. Pour l'instant contentons-nous d'une petite boîte noire permettant l'écriture et la lecture dans un fichier.

Nous devons distinguer l'écriture de texte et l'écriture d'objet.

Pour écrire l'adresse email,

```
final String FICHIER_EMAIL = ".t2e_email.txt" ;
String email = "pbt" ;
FileWriter fw;

try {
    fw = new FileWriter(FICHIER_EMAIL) ;
    fw.write(email) ;
    fw.close();
} catch ( IOException ioe ) {
    ioe.printStackTrace() ;
}
```

Pour lire l'adresse email,

```
final String FICHIER_EMAIL = ".t2e_email.txt" ;
BufferedReader br ;
String s ;

try {
    br = new BufferedReader( new FileReader("FICHIER_EMAIL")) ;
    s=br.readLine() ;
    br.close() ;
} catch ( IOException ioe ) {
    ioe.printStackTrace() ;
}
```

Pour lire / écrire les objets, on utilisera plutôt les " *streams* ". Attention, afin de pouvoir lire / écrire un objet, il doit être *serializable*. Chaque classe voulant être écrite **devra** implémenter l'interface **Serializable**. Les détails seront vu au cours ... à ce stade n'oubliez pas d'ajouter implements Serializable lorsque vous définissez un objet devant être lu / écrit.

Pour écrire un objet,

```
final String FICHIER = ".t2e_todos.dat" ;
MyObject mo ; // un objet quelconque
ObjectOutputStream oos ;

mo = new MyObject("Un_object_quelconque") ;

try {
    oos = new ObjectOutputStream( new FileOutputStream(FICHIER)) ;
    oos.writeObject(mo) ;
}
```

```

oos.close();
} catch ( IOException ioe ) {
    ioe.printStackTrace() ;
}

```

Pour lire un objet,

```

final String FICHER = ".t2e_todos.dat" ;
Object o ;
ObjectInputStream ois ;

try {
    ois = new ObjectInputStream ( new FileInputStream ( FICHER ) ) ;
    o = ois.readObject() ;
    ois.close() ;
} catch ( IOException ioe ) {
    ioe.printStackTrace() ;
} catch ( ClassNotFoundException cnfe ) {
    cnfe.printStackTrace();
}

```

3.2 Envoi d'email

3.2.1 Envoi effectif d'emails, la classe Mailer

Afin de pouvoir envoyer des mails, vous utiliserez le package **Javamail** et la classe **Mailer** que nous vous fournirons. Cette classe définit, entre autres, une méthode `send` permettant d'envoyer un mail ... c'est magique ! Il suffit d'écrire

```

public class TestMailer {
    public static void main ( String[] args ) {
        System.out.println("Test_d'envoi_de_mail");
        Mailer.send("Test", "gxxxx" ) ;
    } // end - main
} // end - classe

```

Par défaut, le mail est envoyé au serveur de mail local (*localhost*) . Si vous n'en avez pas, vous pouvez en installer un⁴ ou utiliser celui de votre provider. Si vous utilisez le serveur de mail de votre provider, il faut modifier le `server` par défaut de la classe **Mailer** en fonction de votre FAI. Ceci se fait grace à la méthode `setServer` de la classe. Pour Coditel, ce sera

```

Mailer.setServer("smtp.coditel.net") ;

```

chez Skynet, c'est

```

Mailer.setServer("relay.skynet.be") ;

```

Pour que ceci fonctionne, il y a deux trois petites choses à mettre en place. En effet, brutalement ça ne fonctionnera pas chez vous.

⁴Les plus connus sont *Postfix* et *Exim*

1. Installer **Javamail**.

Pour ce faire, récupérer le fichier qui va bien chez SUN (<http://java.sun.com/products/javamail/>), dézipper "près"⁵ de votre JDK et ajuster votre CLASSPATH⁶.

2. Pour fonctionner Javamail à besoin de l'API **JAF**.

Récupérer l'archive chez SUN (<http://java.sun.com/products/javabeans/jaf/index.jsp>), dézipper à l'endroit qui va bien⁷ et ajuster le CLASSPATH⁸.

3. Installer le serveur mail ou utiliser celui de son provider.

3.2.2 Objet-singe, la classe **MockMailer**

Un *mock-object*⁹ est un objet permettant les tests. Cet objet mime, simule, ... le comportement d'un objet non encore implémenté par exemple. Dans notre cas vous pouvez utiliser un tel objet, **MockMailer**, afin de tester votre programme.

Avant de mettre en place l'environnement permettant d'envoyer effectivement les emails, vous pouvez utiliser la classe **MockMailer**. Cette classe ne fait strictement rien si ce n'est écrire sur la sortie standard.

Cette classe possède les mêmes méthodes que la classe **Mailer** mais n'exécute pas les mêmes actions. Lors de l'appel de la méthode `setServer`, cette classe écrit un message sur la sortie standard, idem lors de l'appel de la méthode `send`.

Cette classe ne vous servira que dans la phase de développement de votre programme ... afin de pouvoir mieux gérer votre temps.

Webographie

[1] Java Team. Javamail api. <http://java.sun.com/products/javamail/>.

[2] Java Team. Jaf api. <http://java.sun.com/products/javabeans/jaf/index.jsp>.

⁵Si vous avez installé Java dans le répertoire `/usr/local/java/jdk`, vous pouvez placer Javamail dans `/usr/local/java/javamail` par exemple

⁶Ajouter `/usr/local/java/javamail/mail.jar` et `/usr/local/java/javamail/lib/smtplib.jar` dans notre exemple

⁷`/usr/local/java/jaf` par exemple

⁸Ajouter `/usr/local/java/jaf/activation.jar` par exemple

⁹*Mock* se traduit par moquerie, un *mock-object* est traduit par objet factice, simulacre d'objet, objet bidon, objet fantaisie, objet imitation ... je choisis *objet-singe* puisque l'imitation est le propre du singe.

Table des matières

1	Les fonctionnalités	2
1.1	Option <i>new</i>	2
1.2	Option <i>email</i>	2
1.3	option <i>run</i>	2
1.4	option <i>add</i>	2
1.5	option <i>list</i>	2
2	Implémentation	3
2.1	Implémentation de <code>TodoItem</code>	3
2.2	Implémentation de <code>TodoList</code>	4
2.3	Implémentation de <code>Todo2email</code>	4
2.4	Implémentation des tests	4
3	Aspects techniques	5
3.1	Lire et écrire dans un fichier	5
3.2	Envoi d'email	6
3.2.1	Envoi effectif d'emails, la classe <code>Mailer</code>	6
3.2.2	Objet-singe, la classe <code>MockMailer</code>	7