# *TD 4 R2.01*

## Objectifs du TD

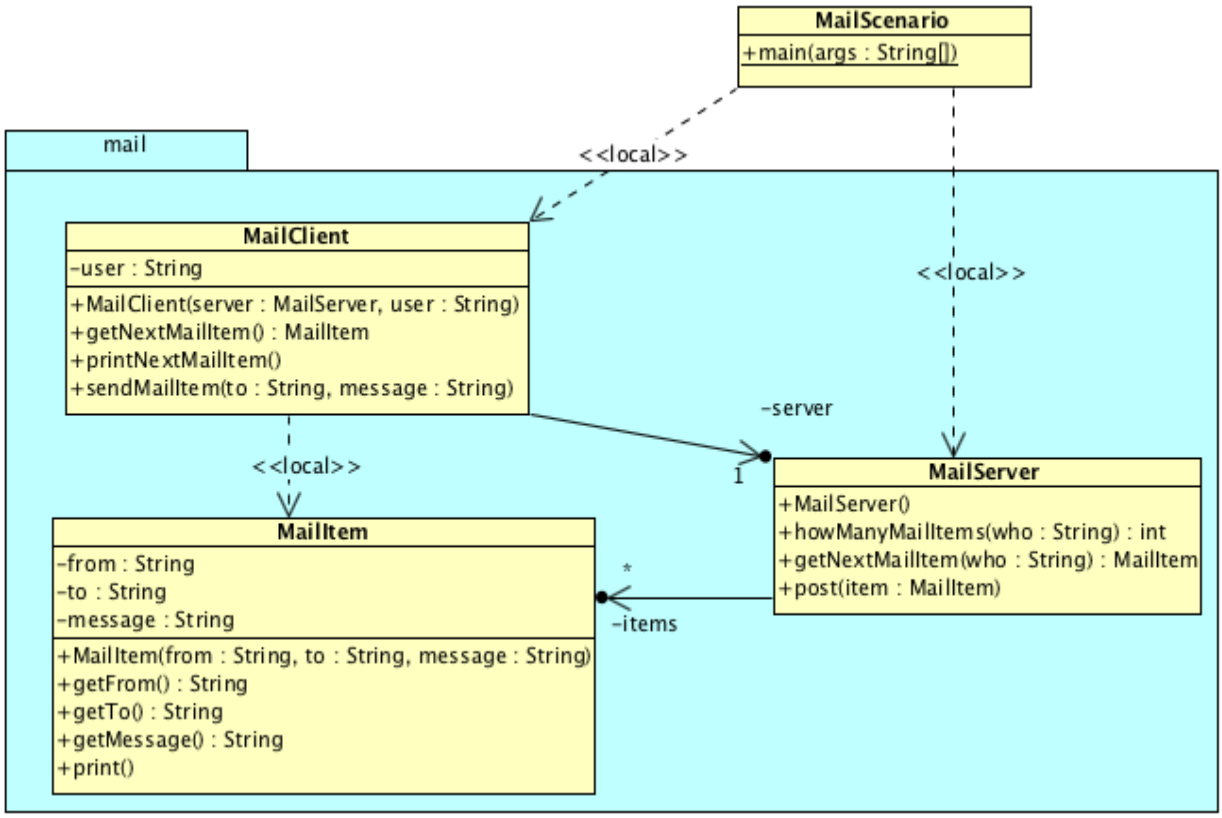
Comprendre et manipuler des interactions entre des objets
Utiliser une nouvelle structure de données : type ArrayList
Utiliser le polymorphisme (méthodes)
Utiliser les packages

## Calendrier

L'application Messagerie contient 3 trois classes et sera complétée par d'autres services, c'est pourquoi nous ne faisons que la première partie cette semaine et nous terminerons la semaine prochaine.

# L'application Messagerie (semaines 7 et 8)

L'application messagerie modélise l'envoi de messages entre des clients de messagerie via l'intermédiaire d'un serveur de messagerie qui gère l'échange de messages.




Le serveur est chargé de recevoir et conserver les messages envoyés par un client à un autre client.

L'application comprend trois classes : **`MailServer`**, **`MailClient`** et **`MailItem`** que l'on laissera en visibilité **`« public ».`**

- Il n'y aura qu'une seule instance du serveur et elle sera utilisée par tous les clients (MailClient). C'est le serveur qui stocke les messages échangés entre les clients.
- Les instances de **`MailClient`** ont un nom d'utilisateur qui leur sert d'identifiant dans les échanges de mail. Les noms des clients doivent donc être uniques.

- Les messages sont envoyés entre les clients via la méthode **`sendMailItem(String to, String message)`** où le paramètre *to* désigne le destinataire du message et *message* le texte du message. .
- Les clients peuvent récupérer leurs messages sur le serveur avec la méthode **`getNextMailItem()`** qui retournera le dernier message enregistré
- La classe **`MailItem`** n'est jamais directement instanciée par un utilisateur. Une instance est créée lors de l'envoi d'un message par un client à un autre.

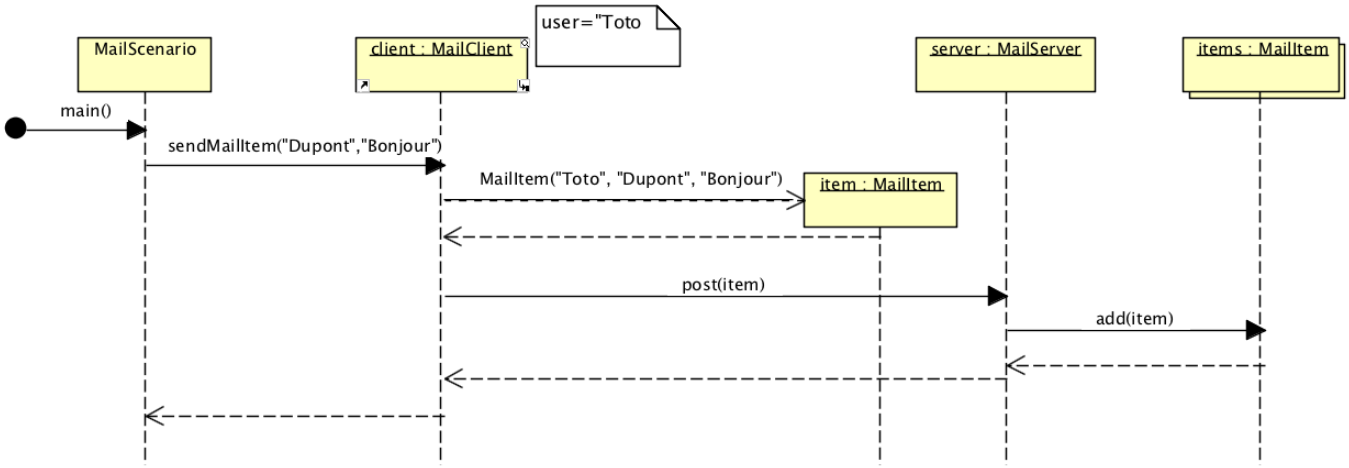
Le diagramme de classes ci-dessus permet de voir les interfaces des classes et leurs relations, on remarquera également que les classes de l'application sont mises dans un package `mail`.
**Voir plus bas les explications des différentes classes.**

## Question 1

Commenter le diagramme de classes et expliquer les différentes relations entre les classes.

## Question 2 : Utilisation de diagrammes de séquence pour comprendre le fonctionnement

Ci-dessous, le diagramme de séquence simplifié montre l'interaction entre les objets pour l'envoi d'un message d'un MailClient nommé " Toto" à un autre nommé " Dupont" et le texte du message est " Bonjour !" .




La classe `MailScenario` n'est utilisée ici que pour indiquer d'où part le premier message.

Expliquer ce que représente le diagramme de séquence en écrivant les messages en Java et dans quelle classe/méthode ils se trouvent.

## Question 3 (à finir en TP)

- Ecrire le code Java de la classe MailItem,
- Commencer à écrire le code Java de la classe MailServer : les méthodes *howManyItems* et *post*

# Explication des différentes classes

## 1- La classe MailItem

Il s'agit d'une classe très simple dont les objets sont des structures de données regroupant les objets qui contiennent le texte du message (*message*), le nom de l'expéditeur (*from*) et celui du récepteur (*to*).

Les méthodes de la classe MailItem sont un constructeur, des accesseurs et une méthode **`print`** pour imprimer le message à l'écran.

Cette classe ne contient pas de modificateur donc une fois un item créé on ne le modifie pas.

## 2 - La classe MailServer

Le serveur utilise une liste de type *ArrayList* pour conserver les messages qui sont des instances de **`MailItem`**.

Il ne possède qu'un attribut : la liste **`items`** d'objets de type MailItem.

**Les méthodes**

- **`MailServer()`** le constructeur qui crée l'ArrayList « items »
- **`post(MailItem item)`** ajoute l'objet de type MailItem passé en paramètre à la liste d'items.
- **`int howManyMailItems(String who)`** retourne le nombre de messages adressés au MailClient dont le nom est passé en paramètre.
- **`MailItem getNextMailItem(String who)`** retourne le premier message (un MailItem) adressé au MailClient dont le nom est passé en paramètre et le supprime de la liste des messages. S'il n'y a pas de message pour le client alors cette méthode retourne null.

## 3- La classe MailClient (à faire semaine 10)

Elle possède deux attributs : une chaîne de caractères pour le nom du client et une référence sur le serveur, c'est-à-dire une instance de **`MailServer`**.

**Les méthodes**

- **`MailClient(MailServer server, String user)`** le constructeur initialise les attributs avec les paramètres.
- **`MailItem getNextMailItem()`** retourne un MailItem en envoyant au serveur le message **`getNextMailItem(user)`** où `user` est le nom du client.
- **`printNextMailItem()`** fait appel aussi au serveur avec le message **getNextMailItem(user)** et imprime le message (méthode print de MailItem) s'il s'agit bien d'un message.
- **`sendMailItem(String to, String message)`** reçoit en paramètre le nom du destinataire et le texte du message, crée une instance de MailItem et envoie le message `post()` avec le mailItem en paramètre au serveur.

Remarque :
Pour que cette application fonctionne on comprendra que tous les objets de type MailClient doivent partager le même objet MailServer.

## 4- La classe MailScenario (à compléter semaine 10)

La classe **`MailScenario`** propose des exemples d'utilisation des différentes classes de l'application en créant un serveur, des clients et des échanges de messages.

# *TP 4 R2.01*

## *Travail à faire en TP (semaine 7)*

- Créer un répertoire *mail* dans votre dossier source pour organiser votre code de manière similaire au dossier *class*.
- Implémenter en Java la classe **`MailItem`** dans le package mail
- implémenter une classe de test `TestMailItem` (tests complets automatisés) à l'extérieur du package mail.
- Implémenter en Java la classe **`MailServer`** dans le package mail
- Générer la javadoc pour les classes **`MailItem`** et **`MailServer`** dans le dossier doc
- Faire une première version de la classe **MailScenario** à l'extérieur du package mail, dans laquelle créer un serveur et des instances de la classe Mail Item. Essayer les différentes méthodes du serveur.

### Travail à rendre :

Déposer sur la zone moodle : votre dossier source et votre dossier doc contenant la javadoc générée