ensiie
école nationale supérieure d'informatique pour l'industrie et l'entreprise

ILO
TP n° 5

Mises à jour

Un certain nombre de mises à jour ont été effectuée dans le code qui vous est fourni. Vous pourrez donc les récupérer dans l'archive /pub/ILO/TP5_update.zip
Les modifications concernent les classes suivantes (que vous pourrez alors copier/coller dans votre projet actuel).

- Ajout d'une classe examples/**TestMessagesStream**.java vous montrant comment utiliser les flux de « Message »s afin de les trier et/ou de les filtrer. Le tri des messages utilise les critères de tri déjà présents dans la classe Message.
- Mise à jour de la classe models/**Message** : les critères de tri sont maintenant stockés dans un Vector et non plus dans un Set afin d'ajouter et/ou d'enlever des critères de tri dans un ordre quelconque. Ce qui sera utilisé dans le nouveau client graphique.
- Mises à jour dans la classe models/**NameSetListModel** afin de préciser les opérations à réaliser.
- Mise à jour de la classe widgets/**AbstractClientFrame** afin d'y ajouter une Map<String, Color> permettant de stocker les couleurs à appliquer aux messages des différents utilisateurs sans avoir à les recalculer en permanence dans la méthode getColorFromName(String name).
- Modification mineure dans la classe widgets/**ClientFrame** dans la méthode parseName permettant de renvoyer la partie « auteur » d'un message texte. Ceci afin que les couleurs à appliquer aux messages des utilisateurs soient les mêmes entre le premier client graphique et le nouveau client graphique.

Précisions sur le nouveau client graphique

- En dehors de la création graphique de votre nouveau client, il est important de répertorier toutes les « actions » que devra réaliser votre client : à chaque action correspond une instance d'une classe héritant de AbstractAction (comme c'est déjà le cas dans le premier client « ClientFrame »).
- Dans le nouveau client graphique, les messages sont transmis au client sous forme d'objets (Message) plutôt que de texte. Vous garderez les messages dans une collection (un Vector par exemple), ce qui permettra de les trier et/ou de les filtrer en utilisant les flux (stream).
    - Le tri d'un flux de messages (tel que présenté dans l'exemple TestMessagesStream) se fait naturellement grâce à la méthode CompareTo déjà présente dans la classe Message. Changer de critère de tri revient donc simplement à changer les critères de tri à appliquer aux messages avec les méthodes de classe Message.addOrder(…) et Message.removeOrder(…).
    - Le filtrage d'un flux de messages suivant leur auteur doit se faire en utilisant un prédicat (Predicate<String>) sur les noms des auteurs des messages. L'exemple TestMessagesStream définit un prédicat « zebulonFilter » permettant de filtrer les messages dont l'auteur est « Zébulon ». Il vous faudra néanmoins construire votre propre classe de prédicat (**AuthorListFilter**) dans laquelle vous devrez pouvoir ajouter et retirer des noms en fonctions des noms sélectionnés dans la liste des utilisateurs (voir la Figure 1 page 2).
- La Liste des utilisateurs doit contenir une liste d'auteurs de messages uniques et toujours triée. La manière la plus simple de procéder est de créer son propre « modèle de liste » sous jacent au

widget JList qui doit afficher cette liste : Vous avez pour ce faire une ébauche dans la classe models/**NameSetListModel** que vous pourrez compléter et utiliser dans votre nouveau client graphique en conjonction avec le « ListSelectionModel » du widget JList pour gérer les éléments sélectionnés ou désélectionnés de la liste des utilisateurs.

- Pour effectivement gérer les sélections d'auteurs dans cette liste et pouvoir l'appliquer au filtrage des messages, vous aurez aussi besoin d'un contrôleur adéquat : Vous aurez donc besoin de créer une classe dans votre client graphique implémentant l'interface « **ListSelectionListener** » que vous associerez à votre JList et qui sera donc appelée à chaque fois que la sélection des éléments changera dans la liste des utilisateurs, vous permettant ainsi de mettre à jour votre « filtre à auteurs », que celui ci soit en application ou pas.
- L'affichage des messages du premier client graphique dans le document sous-jacent à la zone de texte se faisait au fur et à mesure de la réception des messages (le dernier message étant simplement ajouté à la fin du document). Dans le nouveau client graphique, les messages étant conservés dans une collection de messages qui peut éventuellement être filtrée et/ou triée il faudra donc effacer le document et réafficher l'ensemble des messages lorsque les critères de filtrage ou de tri changeront.
- L'action de filtrage des messages est une action binaire et non pas ponctuelle comme les autres actions : elle est en cours d'application ou pas. Il faudra donc l'associer dans la barre d'outils en haut avec un JToggleButton plutôt qu'avec un JButton et dans le menu « Message » avec un JCheckBoxMenuItem plutôt qu'avec un JMenuItem. Par ailleurs, dans la méthode actionPerformed(ActionEvent e) de cette action, vous pourrez caster la source de l'événement e en AbstractButton (JCheckBoxMenuItem et JToggleButton héritent toutes deux de cette classe) afin de déterminer si cette action est sélectionnée (on) ou pas (off). Et enfin, cette action binaire pouvant être associée à plusieurs widgets (un bouton et un item de menu en l'occurrence) il faudra s'assurer que lorsqu'elle est sélectionnée en utilisant le bouton, cette sélection se reflète dans l'item de menu et vice-versa.
- Les différentes actions de tri (par auteur, par date ou par contenu) ne diffèrent entre elles que par l'ordre à appliquer aux messages. Il serait donc pertinent de créer votre propre classe SortAction héritant de AbstractAction que vous pourrez alors instancier 3 fois (avec un ordre de tri différent à chaque fois).

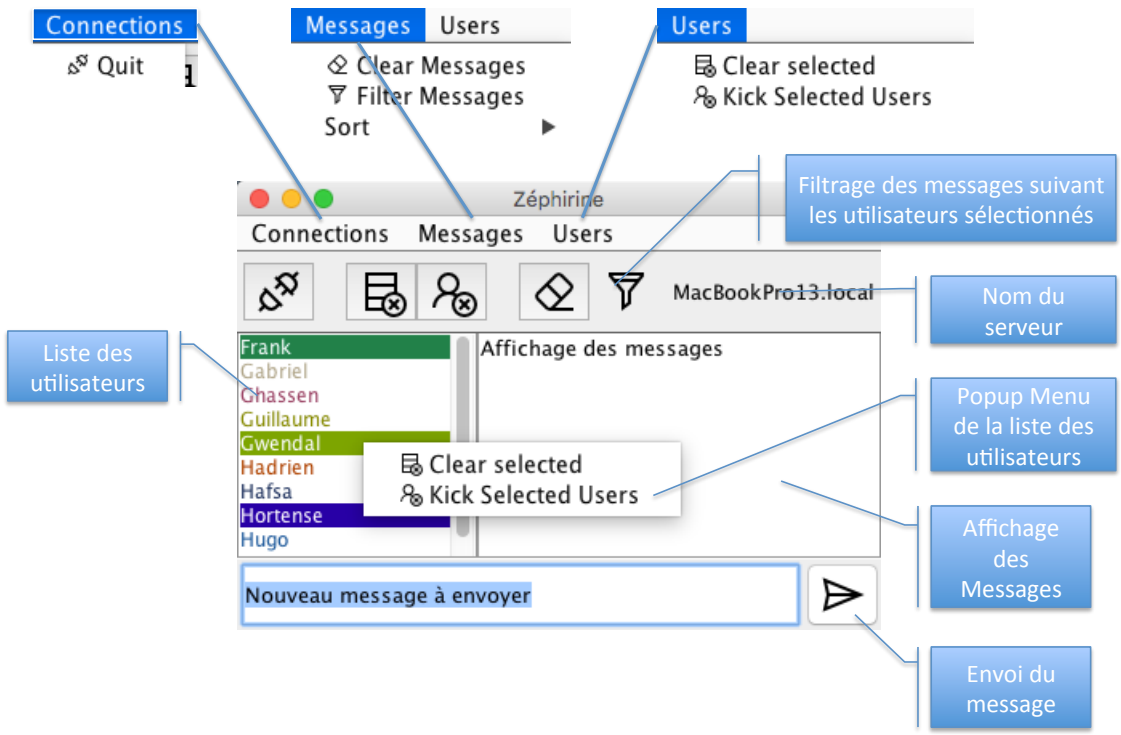

Figure 1 : Aspect du nouveau client graphique