Discussion instantanée : mise en pratique du MVC

L'objectif de ces exercices est de définir une application de discussion instantanée (chat). Bien sûr, nous n'aborderons pas les aspects « application distribuée ». Ainsi, nous nous contenterons d'avoir plusieurs instances lancées depuis le même programme (`ChatSwing`) qui permettent d'ajouter de nouveaux messages dans l'unique salon de discussion (`Chat`).

La figure 1 donne un aperçu de l'interface utilisateur attendue. La partie centrale affiche les messages échangés entre les participants. La zone inférieure rappelle le pseudo de l'utilisateur, une zone de saisie permet d'entrer le texte et le bouton OK l'envoie sur le chat. Le message (pseudo + texte) s'ajoute alors dans la partie centrale de cette fenêtre et sur toutes les autres vues sur le même chat.

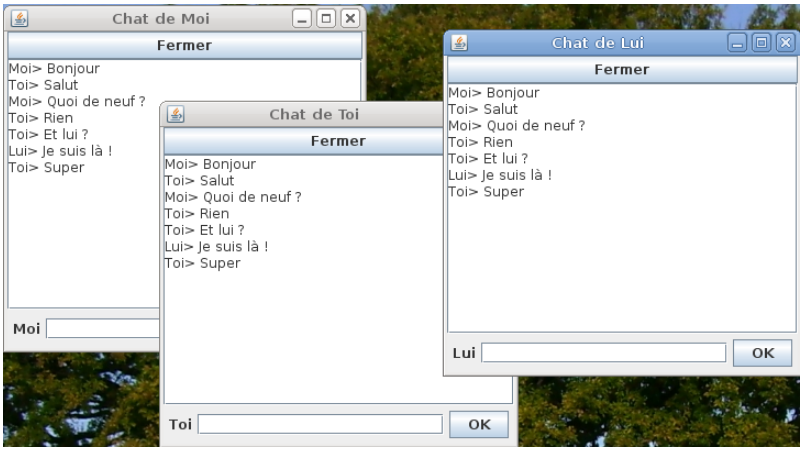

FIGURE 1 – L'interface utilisateur pour accéder au chat

La partie supérieure contient un bouton « Fermer » qui fait disparaître la fenêtre et donc correspond à la déconnexion de l'utilisateur. Les fenêtres correspondant aux autres utilisateurs restent actives et ils peuvent continuer à dialoguer.

Le principe est de créer autant d'instances de la classe `ChatSwing` que d'utilisateurs souhaitant discuter dans un salon de discussion. Lorsque l'un des utilisateurs envoie un nouveau message, il est reçu par tous les utilisateurs du salon de discussion, lui compris. Nous nous appuyons sur le patron MVC avec modèle actif pour mettre en place ce système.

Exercice 1 : Le modèle Chat

La classe `Chat` (listing 1) propose une première version simplifiée du modèle. Le chat consiste simplement à enregistrer dans une liste les messages.

1.1. Expliquer pourquoi l'attribut `messages` est déclaré du type `List` mais initialisé en utilisant la classe `ArrayList`.

Listing 1 – La classe Chat

```
1  import java.util.List;
2  import java.util.ArrayList;
3  import java.util.Collections;
4
5  public class Chat {
6
7        private List<Message> messages;
8
9        public Chat() {
10             this.messages = new ArrayList<Message>();
11       }
12
13       public void ajouter(Message m) {
14             this.messages.add(m);
15       }
16 }
```

1.2. Expliquer pourquoi ce serait une mauvaise idée d'avoir un accesseur `getMessages()` qui retourne la valeur de l'attribut `messages`.
1.3. Modifier la classe `Chat` de sorte que l'on puisse faire un `foreach` sur une variable de type `Chat` et ainsi obtenir successivement chacun des messages enregistrés dans le `Chat`.
1.4. Utiliser le patron Observateur pour rendre ce modèle actif. Dès qu'une modification est faite (ajout d'un nouveau message), les observateurs inscrits auprès du chat sont mis à jour. On utilisera la classe `Observable` et l'interface `Observer` du paquetage java.util.
1.5. Écrire un observateur qui affiche sur la sortie standard le dernier message ajouté au chat.

Exercice 2 : Composant VueChat
Intéressons nous à la vue graphique, la classe `VueChat`, présentant les messages à l'utilisateur. Il s'agit de la partie centrale sur la fenêtre de la figure 1. Ce composant graphique aura pour modèle une instance de `Chat` et sera mis à jour dès qu'un nouveau message est ajouté au modèle.
2.1. Écrire la classe `VueChat`.
2.2. Compléter le programme principal précédent pour visualiser graphiquement les messages. On pourra définir une classe `VueChatFenetre` qui crée la JFrame contenant la vue.

Exercice 3 : Définition du contrôleur ControleurChat
La partie inférieure de la fenêtre (figure 1) correspond au contrôleur. Il permet à l'utilisateur de saisir un message et de le diffuser à tous.
3.1. Définir la classe `ControleurChat` correspondant au contrôleur.
3.2. Compléter le programme principal pour y intégrer le contrôleur. On pourra définir une classe `ControleurChatFenetre` qui intègre le composant (`ControleurChat`) dans une `JFrame`.

Exercice 4 : La classe ChatSwing
Définir une classe `ChatSwing` qui, comme sur la figure 1, intègre à la fois la vue et le contrôleur déjà définis.