TD-TP : Le Design pattern « Médiateur »

Nous avons vu que les principes SOLID tendent notamment à diminuer le couplage entre modules.
Le Patron de Conception Médiateur relève de cette préoccupation.

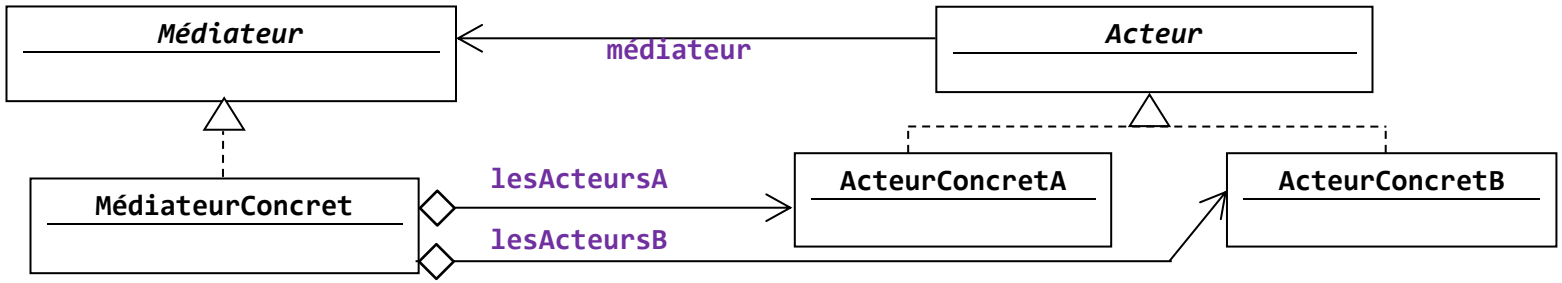

Schéma de classe global du Patron de Conception « Médiateur »

On peut illustrer la mécanique du Patron de Conception Médiateur par la « *régulation d'un aéroport* » :

- Les différents **`ActeursConcrets`**, que sont les **`avions`** et **`hélicoptères`**, ne sont pas en communication directe entre eux pour accéder à l'aéroport.
- Ils passent par un **`MédiateurConcret`** qu'est la **`tourDeContrôle`** et qu'ils connaissent tous.
- Les informations sont centralisées par la **`tourDeContrôle`** qui coordonne les atterrissages et décollages **`lesActeurs`** que sont les **`avions`** et les **`hélicoptères`**.
- Dans ce cas de *« régulation d'un aéroport »* nous avons donc deux familles d'acteur concrets : les **`avions`** et les **`hélicoptères`**.

Travail à faire : Le **`Médiateur`** et les **`Acteurs`** se connaissent

Un autre exemple d'usage du Patron de Conception Médiateur est celui d'un « *chat* » sur lequel plusieurs utilisateurs échangent :

- Tous les **`utilisateurs`** du « *chat* » sont des **`ActeursConcrets`**, et aucun n'est en communication directe avec les autres utilisateurs.
- Chaque **`utilisateur`** est en communication directe avec **`unServeurDeChat`**, qui joue le rôle de leur **`MédiateurConcret`**.
- Les messages sont reçus par **`unServeurDeChat`** qui les diffuse à l'ensemble de **`lesActeurs`** que sont les **`utilisateurs`**.
- Dans le cas du « *chat* » nous n'avons donc qu'une seule famille d'**`ActeursConcrets`** ; ce sont les **`Utilisateurs`**.

1. Etant donné le schéma de classe global du Médiateur et les caractéristiques du « *chat* » données ci-dessus, représentez le schéma de classe (objets inclus) qui modélise la situation d'un « *chat* » où l'objet **`unServeurDeChat`** a en charge de gérer la communication de trois utilisateurs, **`utilisateur1`**, **`utilisateur2`** et **`utilisateur3`**, chacun caractérisés par un **`String nom`**.

   On attend que vous fournissiez :
   - les classes,
   - les relations entre ces classes,
   - les attributs et méthodes qui implémentent ces relations,
   - les 4 objets considérés.

2. Donner le code des méthodes ajouter/retirer/existe de la classe **`ServeurDeChat`**, propres à la gestion de ses utilisateurs (cf. ses acteurs).

3. Donner le code d'un **`main()`** qui crée les 4 objets et lie le serveur de chat (cf. le médiateur) à ses 3 utilisateurs (cf. les acteurs).

Travail à faire : Un **`Médiateur`** communique avec ses **`Acteurs`** (mise en œuvre de la dynamique pour la situation de *« chat »*)

Sachant que pour la situation *« chat »* :

- le **`ServreurDeChat`** propose dans son interface la méthode **`void diffuserMessageDe (String message, Utilisateur auteur)`** qui permet à un **`Utilisateur`** de se signaler au **`ServeurDeChat`** dès qu'il souhaite communiquer un message sur le Chat, et que
- le code de cette méthode consiste pour le **`ServeurDeChat`** à diffuser l'information auprès de chacun des utilisateurs, dont il a la charge, en invoquant la méthode **`void recevoirMessageDe (String message, Utilisateur auteur)`** définie dans l'interface **`Utilisateur`**

4. Complétez la solution de votre schéma de classe UML précédent

5. Nommez chacune des classes concernées par la mise en œuvre de cette dynamique et pour chacun d'elles, donnez le code des méthodes à intégrer au projet, sachant que **`void recevoirMessageDe (String message, Utilisateur auteur)`** se caractérise par un affichage du message reçu, et du nom de son émetteur.

6. Compléter le **`main`** de sorte à ce que chaque utilisateur salue les acteurs avec lesquels il *« chat »* via le serveur de chat.

7. Quel sera le résultat/affichage obtenu ?