Design Patterns - Synthèse

Axel Gendillard
Audoin De Chantérac

Sommaire

1 Design Pattern - Singleton

1.1 Objectifs

Les objectifs de l'incrément 1 sont les suivants :

— Créer un nouvel utilisateur à chaque validation ;
— Avoir un utilisateur unique par login ;
— Créer plusieurs terminaux par utilisateur ;
— Interdir la création d'utilisateur hors de la fenêtre.

Pour répondre à ces objectifs nous allons utiliser le design pattern Singleton. En effet, le Singleton garantie l'unicité de l'intanciation de la classe User.

1.2 Implémentation

Dans le but d'implémenter notre solution, nous avons crée dans la classe User la méthode getInstance() : public synchronized static User getInstance(String login, String password). Cette méthode est static, ce qui permet d'avoir une instance unique de cette classe. Cette méthode est publique, ce qui donne un point d'accès universelle à une instance donnée.
Le mot clé synchronized permet de gérer les problématiques d'accès concurrent dans le cas de programmation multitâche. On s'assure ainsi en effet que la ressource ne sera pas partagé à un instant du programme par plusieurs tâche.

La méthode getInstance() permet à partir d'un login et d'un mot de passe passé en paramètre de retourner :

— L'instance existante de user si le login est déjà connu ;
— Retourne une nouvelle instance de la classe user après avoir ajouté le login dans une liste d'instance (servant de mémoire).

2 Design Pattern - Observer

2.1 Objectifs

Les objectifs de l'incrément 2 sont les suivants :

— Synchroniser les historiques des terminaux ;
— Mettre à jour tous les terminaux lors de l'entrée d'une commande.

L'API Java nous fournit déjà l'interface Observer et la classe Observable.

2.2 Implémentation

Pour commencer, nous avons actualisé les notions d'observer et d'observables. Les instances "observers" sont les instances de la classe VirtualOS. Nous avons implémenté l'interface java.util.Observer dans la classe VirtualOS.

Les instances "observables" sont les instances de la classe User. Nous avons étendu la classe java.util.Observable dans la classe User. Cette classe est liée à un historique (classe Historic). La classe Historic a une liste de commandes et est liée à un utilisateur (classe User).

Afin de gérer la synchronisation des historiques lors de l'ouverture d'un nouveau terminal, nous avons créé une méthode : copyCommand(). Cette méthode parcours la liste des commandes du terminal courant et reproduit les entrées et sorties de l'utilisateur dans le nouveau terminal. Cette méthode est appelé dans le constructeur de TerminalOS.

Afin de gérer la synchronisation dynamique des terminaux, nous avons utilisé l'interface Observer et la classe Observable.

L'interface Observer n'a qu'une seule méthode : void update(Observable o). Dans virtualOS, on a "override" cette méthode en ajoutant les chaînes de caractères que nous voulons mettre à jour. On récupère seulement la dernière commande de l'historique : getLastHistoric().

L'appel de la méthode handleCommand, on effectue la mise à jour des terminaux par l'appel de addCommand. Afin d'éviter une mise à jour du terminal courant, on supprime ce terminal dans la liste des observers (deleteObserver). On le réajoute après la synchronisation (addObserver).

La classe User a une méthode permettant l'ajout d'une commande dans la liste et la synchronisation des terminaux : addCommand(String command). Cette méthode utilise les méthodes de la classe Observable pour la synchronisation : setChanged() (permet la modifiction des terminaux via un booléen contenu dans la classe Observable : changed) et notifyObservers() (appelle la méthode update de l'interface Observer afin de mettre à jour les terminaux).

3 Design Pattern - Visitor

3.1 Objectifs

Les objectifs de l'incrément 3 sont les suivants :

— Interpréter des commandes du terminal (ls et cat) pour chaque type de noeud ;
— Utiliser le patron Visitor.

3.2 Implémentation

4 Design Pattern - Command

4.1 Objectifs

Les objectifs de l'incrément 4 sont les suivants :

— Interpréter des commandes du terminal (LS et CAT) pour chaque type de noeud ;
— Utiliser le patron Visitor.

4.2 Implémentation