M. Gasquet, N. Lazaar, C. Nadal, N. Palleja, S. Robillard

Patrons État et Commande

1 Patron État

Nous souhaitons réaliser une simulation simpliste d'une socket. Celle-ci possède les états : `"initial"`, `"écoute"`, `"connectée"`, `"fermée"`. Les méthodes suivantes sont définies ainsi :

- `listen()` fait passer de l'état `"initial"` à l'état `"écoute"`.
- `read()` affiche un message dans la console uniquement dans `"connectée"`.
- `close()` fait passer de l'état `"écoute"` ou de l'état `"connectée"` à l'état `"fermée"`.
- `connect()` fait passer de l'état `"initial"` à l'état `"connectée"`.
- `accept()` fait passer de l'état `"écoute"` à l'état `"connectée"`.

Question 1 • Donnez le diagramme états-transitions de l'entité socket.

Question 2 • Proposez une conception UML antipattern de cette simulation.

Question 3 • Implémentez votre conception en Java.

Question 4 • Proposez maintenant une seconde conception UML en utilisant le patron *État*.

Question 5 • Implémentez votre seconde conception en Java.

Nous souhaitons maintenant enrichir le système en ajoutant la transition `reset()` qui permet de passer de l'état `fermée` à l'état `initial`. Nous souhaitons également ajouter un état intermédiaire entre `écoute` et `connectée`, nommé `configuration`, qui permet de configurer la connexion de la socket. Désormais, un `accept()` permet de passer de l'état `écoute` à l'état `configuration`. l'activité `config()` est lancée uniquement dans l'état `configuration`, une fois la configuration terminée, on passe automatiquement à l'état `connectée`.

Question 6 • Ajoutez cette nouvelle extension à vos deux conceptions/réalisations et analysez les principes SOLID.

Question 7 • Implémentez des tests qui permettent de visiter chaque état 10 fois.

Question 8 • Récupérez et analysez le fichier log du GC en relançant vos tests avec comme argument JVM : `-Xlog:gc*:file=<PATH_TO_GC_LOG_FILE>`.

2 Patron Commande

Dans cette partie, nous souhaitons gérer l'ensemble des opérations appliquées sur les sockets comme des commandes.

Question 9 • Donnez le diagramme de classe qui permet d'utiliser le patron Commande dans notre étude.

Question 10 • Créez une classe simple appelée `Simulator` qui possède les méthodes pour effectuer des opérations sur les sockets.

Question 11 • Créez une interface appelée `Commande` qui possède une seule méthode `executer`.

Question 12 • Créer des classes concrètes qui implémentent l'interface `Commande` avec les commandes de **Version1**. Ces classes doivent avoir des champs privés pour l'objet `Socket` et doivent implémenter la méthode `executer()` pour effectuer l'opération sur la socket en question.

Question 13 • Ajoutez à la classe `Simulator` les méthodes `setCommand` et `executeCommand`.

Question 13 • Apportez à votre conception et à votre réalisation la modification suivante : `connect()` et `reset()` sont des commandes annulables.

Question 13 • Considérez maintenant la commande `fermee()` comme commande annulable. Vérifiez bien que les principes SOLID sont bien respectés.

Question 14 • Dans la classe `Client`, ajoutez une méthode principale et créez aléatoirement et de manière dynamique une séquence qui permet de visiter chaque état de la socket au moins 10 fois. Analysez de nouveau le log du GC.