Serveur Concurrent

Faites votre propre Pattern!

Objectif :

- **Appliquer l'approche SOLID pour concevoir un patron ou architecture générique du code source**

Chargez le corrigé fourni https://github.com/cours-genie-logiciel-ensta/conception-logicielle-td2-motif-strategie-corrige.git

Cahier des charges

Un nouveau service a été implanté pour permettre au client de la banque d'afficher l'historique de ses opérations. Deux classes nouvelles ont été ajoutées, Historique et HistoriqueGUI et bien sur la classe ServeurSpecifique a été modifiée.

Evaluez les limites de la proposition au regard des hypothèses suivantes :

Supposons (1) que la banque soit utilisable par plusieurs utilisateurs à la fois, et (2) qu'elle puisse proposer de nombreux services en plus du service existant de depot/retrait et de consultation d'historique. Des exemples : dépot de cheque, consultation des autres comptes, demande de modification des autorisations de decouvert, etc.

Trois problèmes se posent :

- A l'utilisation : le serveur chargé de réaliser tous ces services risque d'être saturé par les demandes des utilisateurs. De plus, ces services ne requièrent pas forcément les mêmes ressources (temps de réaction, mémoire, etc.).

- A l'implémentation : si de nouveaux besoins de services apparaissent, l'application, dans son architecture actuelle, risque de devenir de plus en plus difficile à maintenir.

- A l'exécution: l'ajout d'un nouveau service nécessite la modification et le redémarrage de tous les services existants. Pourquoi est-ce le cas?

Par conséquent il faut réorganiser l'application côté serveur qui n'est pas modulaire. Par exemple, on peut avoir plusieurs serveurs (TCP), chacun dédié à un type de service (i.e. *opération*).

Votre travail

A partir de ces explications, vous imaginerez un pattern « serveur concurrent » permettant de créer des architectures logicielles reposant sur l'utilisation de ce pattern.

La banque ne sera qu'un exemple d'utilisation de ce pattern donc le pattern de doit pas tenir compte de la banque.

Pour vous guider, cherchez à appliquer l'approche SOLID avec en priorité le S, I et D. Sachant que, bien sûr, le O et L sont à prendre en compte.

- **Première étape :** Cherchez à appliquer le principe I pour l'inversion de dépendances entre le serveur et la banque ?
    - **Objectif : casser la dépendance entre les deux classes**
        - Le serveur ne doit plus référencer une banque
        - Pour vous orienter, il faut créer une interface.
        - Modifier le code et exécuter l'application.

- **Deuxième étape,** une fois la connexion établie dans le serveur, tous les échanges entre le client et le serveur sont localisés dans la méthode ***run*** de la classe ***ServeurSpecifique***.
    - **Est-ce gênant ? Oui !!!** tous les protocoles entre le client et le serveur sont mélangés alors que les services sont vraiment disjoints.
    - **Objectif : séparer les protocoles**
        - Appliquer le principe S : Une classe et une responsabilité !! Une classe pour un protocole
        - Programmez une solution avec les deux protocoles séparés implantés dans une classe pour chaque protocole.
        - Exécutez votre code.

- **Troisième étape,** cherchez à appliquer le principe I pour l'inversion de dépendances entre le serveur et les différents protocoles implantés
    - **Objectif : casser la dépendance entre les deux classes**
        - Pour vous orienter, il faut créer une interface.
        - Appliquer le pattern Stratégie
        - Modifier le code et exécuter l'application.

- Vous avez réalisé un super boulot de restructuration de votre code source. Il faut donc le capitaliser et décrire un motif de conception générique pour les serveurs concurrents.
    - **Proposez donc un patron de conception, et déposez le sur Moodle,** en sélectionnant les bonnes classes pour qu'elles soient réutilisables dans toutes les applications de serveur concurrent. Vous en aurez besoin dans le projet.
    - **Décrire le patron avec :**

- **un nom**
- **un contexte d'utilisation**
- **un diagramme de classe UML et un diagramme de séquence présentant le fonctionnement pour la banque.**
- **Le code source de la solution**

- **Si vous avez encore du temps :** Si vous avez réalisés votre pattern de client/serveur, vous pouvez améliorer votre description de protocole. Un protocole est implanté à partir d'un automate coté client et serveur. Coté serveur, à partir des classes protocole réalisés, intégrez un Pattern State pour forcer d'effectuer d'abord un dépôt puis un retrait. Les demandes de retrait et dépôt seront effectuées à partir des états internes du serveur. Nous ne pouvons faire un dépôt que si le serveur est dans un état StartDepot (pour imposer le séquencement du dépôt puis du retrait) puis l'automate passe dans un état StartRetrait pour effectuer uniquement un retrait. Dans l'état StartDepot, une demande de retrait de déclenche aucune action. Pour faire ce travail, il faut :
  - Regardez l'implantation du code du Stat Pattern fournie en cours, exécutez là pour bien comprendre.
  - Dessinez l'automate que vous voulez implanter.
  - Créez les classes d'implantation du pattern State et intégrez les.