TP Injection de dépendances

Mise en place

Après avoir cloné le projet github https://github.com/Ninja-Squad/tpdi.git, commencez par éditer la première ligne du fichier gradlew.bat afin de renseigner l'endroit, sur votre disque dur personnel, où gradle devra stocker ses artefacts (gradle lui-même, ainsi que les librairies utilisées).

Lancez ensuite la commande suivante :

```
gradlew eclipse
```

Cette commande va commencer par télécharger Gradle, puis va initialiser un projet Eclipse dans le répertoire courant, que vous pourrez ensuite importer dans Eclipse.

Etape 1

Nous allons construire une mini-application permettant de consulter et créer des spectacles. Cette application sera une application en ligne de commandes.

Le programme devra simplement lire les commandes de l'utilisateur, et y répondre, en lisant les spectacles à partir d'un simple fichier texte, contenant un titre de spectacle par ligne. Les commandes possibles seront :

- **`trouver <chaine>`** : devra lire le fichier, ligne par ligne, et afficher à l'écran tous ceux dont le titre contient la chaine passée en argument
- **`creer <titre>`** : devra vérifier qu'un spectacle ayant le même titre n'existe pas déjà, puis ajouter le titre à la fin du fichier. Une ligne de log devra en outre être écrite dans un autre fichier, contenant le titre du spectacle créé, et le moment où ce spectacle a été créé.
- **`quitter`** : devra terminer le programme

Vous êtes libres d'implémenter ce programme comme bon vous le semble.

Réflexion

Comment pourriez-vous tester ce programme ? Comment pourriez-vous automatiser ces tests ? Pourriez-vous facilement changer la manière dont les spectacles sont lus et stockés sans réécrire tout le programme ?

Etape 2

Plusieurs responsabilités se dégagent de ce programme :

- analyser les commandes et afficher les résultats (controller)
- répondre aux commandes (service)
- lire et écrire les fichiers (DAO – Data Access Object)

Réécrivez votre programme en isolant ces 3 responsabilités dans trois classes différentes.

Réflexion

Le code vous semble-t-il plus facile à comprendre et à maintenir écrit de cette manière ? Pourriez-vous à présent tester plus facilement le code produit ? Pourriez-vous plus facilement changer la manière de lire et stocker les spectacles ?

Etape 3

Injectez manuellement les dépendances de chaque composant pour rendre le code testable. Testez unitairement le controller et le service en utilisant JUnit et Mockito.

Etape 4

Nous allons simuler des transactions. Chaque appel du controller au service devra démarrer une transaction, qui sera committée ou rollbackée après l'appel au service.

La simulation se fera simplement en affichant « début de transaction » à l'appel du service, et « commit de transaction » après l'appel du service. Si le service lève une exception, au lieu du message de commit, on affichera « rollback de la transaction », et on propagera l'exception initiale.

Implémentez ce programme comme bon vous le semble.

Réflexion

Trouvez-vous le code des méthodes du service faciles à lire ? Comment pourrait-on garder cette fonctionnalité, tout en gardant le code fonctionnel tel qu'il était initialement ? Quel design pattern pourrait être mis en place ?

Etape 5

Mettez en place un proxy pour isoler la gestion des transactions dans une classe distincte. Pour cela :

- Créez une interface déclarant les méthodes offertes par le service
- Faites en sorte que la classe du service implémente cette interface
- Créez une classe ServiceProxy implémentant également cette interface. Ce proxy contiendra la gestion des transactions, et déléguera à une autre instance de l'interface (qui sera donc une instance de la classe Service, qui ne contiendra plus la gestion des transactions)

Réflexion

Le code de gestion des transactions n'est-il pas répétitif ? Pourrait-on éviter d'écrire ce code encore et encore, pour chaque service ? Comment ?

Etape 6

Utilisez Spring pour injecter les composants les uns dans les autres.

Etape 7

Copiez-collez les sources du TP sur JPA dans votre projet. N'oubliez pas le fichier persistence.xml, qui devra être placé dans src/main/resources/META-INF. Utilisez le fichier AppConfig.java fourni pour configurer l'application. Vous devrez y ajouter les annotations nécessaires. L'EntityManager pourra être injecté dans le DAO en utilisant l'annotation `@PersistenceContext` sur le champ (ou sur un setter) de type `EntityManager`.

Refactoriser le programme afin de lire et créer des spectacles (avec leur titre, mais aussi leur type et leur artiste). Maintenez les tests unitaires.