TD 4 : Application domotique

Modèles de conception (Fabriques, Adaptateur, Décorateur)

Le site de l'organisation du cours, où vous allez trouver le projet à forker (TD4) :
https://github.com/IUTInfoMontp-M3105

Consignes : Respecter les principes SOLID.

Date limite de rendu sur votre dépôt GitHub privé : **Dimanche 13 Octobre, 23h00**

Vous allez écrire une petite application de domotique. On lui connectera des dispositifs électroniques et de démarrer à distance les dispositifs connectés. Dans tous les cas, vous allez simuler les différentes actions des dispositifs connectés par des messages d'affichage appropriés : par exemple si une radio est démarrée, vous vous contenterez d'afficher "*La radio ... démarre*" ; si elle est configurée avec une certaine fréquence vous afficherez "*La radio et configurée avec la fréquence ...*" ; etc.

Exercice 1 - création des dispositifs

Deux classes vous sont données : `AppliDomotique` et `Connectable`. Dans un premier temps, le problème est de connecter des dispositifs à l'application `AppliDomotique`. Celle-ci gère une collection d'objets de type `Connectable`, qui sont connectés (et donc stockés dans la collection) à la demande de l'utilisateur. Pour le début, on considère deux catégories d'objets possibles : des *radios* et des *cafetières*.

Tout objet `Connectable` possède une méthode `void configurer(String config)` qui permet de configurer le dispositif. On simulera cette action avec des simples affichages propres à chaque dispositif `Connectable`. Par exemple pour la cafetière afficherez le message "*On configure la cafetière avec* `config`", où `config` est la température appropriée. Pour la radio vous afficherez le message "*On configure la radio avec* `config`", où `config` est la fréquence radio préférée.

1. Écrivez deux classes `Radio` et `Cafetière` implémentant l'interface `Connectable` qui vous est donnée. Complétez le corps de leurs méthodes `configurer(String config)` et `demarrer()`.

2. L'application principale `AppliDomotique`, demande en boucle à l'utilisateur de saisir une chaîne de caractère correspondant au type de l'objet connectable qu'il souhaite créer. Voici la procédure de connexion de tout nouveau dispositif connectable :
   - créer l'objet,
   - une fois l'objet connectable créé, invoquer sa méthode de configuration,
   - l'ajouter à la collection des objets connectés.

   Complétez le code de `AppliDomotique` pour réaliser cela.

3. Quels sont les défauts de cette solution si on ajoute un nouveau type d'objets à connecter ? Et si on ajoute une deuxième cafetière parce qu'on boit vraiment beaucoup de café ?
   Discutez avec votre enseignant.

Exercice 2 - création organisée

Afin d'améliorer la solution précédente et pouvoir connecter davantage de catégories d'objets, on va isoler, dans l'application précédente, le traitement qui consiste à créer et configurer les objets : on acceptera des objets de toutes sortes, à condition toutefois que ceux-ci soient `Connectable`.

1. Écrivez une classe abstraite `FabriqueConnectable` proposant trois méthodes :
   - **abstraite** `Connectable creer();`
   - **abstraite** `void configurer(Connectable c);`
   - **concrète** `Connectable fabriquer()` qui *crée* on objet `Connectable`, le configure et le retourne au programme appelant.

2. Pour chaque dispositif connectable écrivez une classe qui hérite de `FabriqueConnectable` et qui s'en chargera de la création du dispositif correspondant. Par exemple pour la radio, vous allez créer la classe `FabriqueRadio`. Implémentez les méthodes de chacune des ces fabriques de manière correspondante.

3. Modifiez la méthode `static void connecter(String type)` de la classe `AppliDomotique` afin de simplifier la création des objets connectables.

4. Proposez une nouvelle classe connectable, `Radiateur`. Est-ce que le processus de connexion (création + configuration du dispositif) est modifié ?

Exercice 3 - adaptation des interfaces

On s'intéresse maintenant à une nouvelle catégorie d'objets : les imprimantes. Il se trouve que ces dispositifs **ne sont pas connectables** à l'application domotique car la classe `Imprimante` a déjà été écrite par quelqu'un d'autre et vous n'avez pas accès au code source pour la modifier... Mais comme toute classe `Java`, vous pouvez l'introspecter pour voir sa déclaration : depuis Intellij IDEA, clic sur le nom de la classe $\longrightarrow$ `CTRL + B`. Vous remarquerez qu'`Imprimante` a une méthode `imprimer()`, qui affiche un message approprié.

1. Même si elles ne sont pas `Connectable`, on veut pouvoir connecter les imprimantes à l'application domotique. Proposez une solution pour adapter l'imprimante, afin qu'elle puisse être connectée et que la méthode `demarrer()` corresponde à la méthode `imprimer()`
2. Vérifiez dans le programme principal en fabriquant l'imprimante et en la démarrant.

Exercice 4 - extension flexible

On souhaite maintenant étendre la classe `Imprimante`. Il peut exister deux sortes d'imprimantes : *locales* ou *en réseau*. Toute imprimante peut aussi être *multifonction* c'est-à-dire avec une ou plusieurs des fonctions suivantes : *scanner*, *fax* ou *photocopieur*.

1. Réfléchissez à un diagramme de classes envisageant toutes les classes possibles et appréciez la complexité.

2. Définissez deux classes `ImprimanteLocale` et `ImprimanteReseau`, héritant toutes les deux de `Imprimante`. Redéfinissez leurs méthodes `imprimer()`. Rappel : il s'agit simplement d'afficher un message appropriée distinct pour chacune des trois classes.

3. Définissez les classes `Scanner`, `Fax` et `Photocopieur`. Implémentez leurs méthodes `imprimer()`. En plus du traitement correspondant au sorte d'imprimante, la méthode `imprimer()` doit afficher le message "*La fonction ... (par ex. Fax) est activée*".

4. Vérifiez votre solution en créant différentes imprimantes qui combinent les divers sous-ensembles de fonctionnalités.