TP 4: Design Patterns (1)

Récupérer l'archive:
http://mathieuacher.com/teaching/MDI/DP.zip

Exercice 1. Distributeurs de bonbons

Dans le dossier « DP », il y a un dossier « bonbons » qui contient l'implémentation d'un programme Java.

Question #1: Que fait ce programme ? Quels sont les défauts de conception ?

Question #2: Spécifier le diagramme de la machine à état du distributeur de bonbon

Question #3: Refactoriser le code en appliquant un design pattern vu en cours

Question #4: On souhaite ajouter un état « gagnant » : le cas se produit lorsque l'utilisateur de la machine tourne la poignée (en ayant préalablement insérer une pièce évidemment) et de manière aléatoire il gagne parfois trois bonbons.

- Décrire rapidement comment vous feriez dans le code d'origine
- Implémenter ce nouvel état à l'aide du même design pattern (cf Question 3)
- (option) Etender le programme pour permettre la paramétrisation du (1) nombre de bonbons gagnés et (2) de la fréquence de gain
- Discuter des avantages et inconvénients de l'application du design pattern (versus le code à l'origine)

Question #5 (option): Ecrire des cas de tests. Fonctionnent-ils sur la version d'origine ?

Exercice 2. Swing et Facture

Question #6:
Faître un programme qui montre comment en Swing/AWT un JPanel peut contenir des JPanels, des JButton et des JLabels

Question #7: Quel est le design pattern utilisé ? Identifier les « rôles » de chaque classe dans le design pattern (n'hésiter pas à lire la documentation de Swing/AWT)

Une société vend des articles de papeterie. On se limitera aux articles suivants :

- stylos décrits par une référence, un nom (par exemple "stylo noir B2"), une marque, un prix unitaire et une couleur,

- ramettes de papier décrites par une référence, un nom, une marque, un prix unitaire et le grammage du papier (par exemple, 80 g/m²).

Cette société peut aussi vendre ces articles par lots.
Une commande est un ensemble d'articles commandés par un client (avec pour chaque article la quantité).

On souhaite implémenter un outil permettant de passer des commandes pour des fournitures de bureau et de sortir une facture pour chaque commande.
(L'objectif n'est pas de construire une interface graphique complète, mais simplement d'implémenter les classes Java nécessaires pour, e.g., formuler des commandes et calculer le prix d'une commande)

Question #8.
Dans un premier temps, on suppose que les lots sont composés d'un certain nombre d'un même type d'articles, par exemple un lot de 10 stylos noirs B2 de la marque Bidule. Un lot a une référence et une marque (celle de l'article).
Le nom du lot est déterminé par le nombre d'articles dont il est composé ; par exemple "Lot de 10 stylos noir B2". Le prix du lot est déterminé par le prix unitaire de l'article multiplié par le nombre d'articles, auquel est enlevé un certain pourcentage qui dépend du lot. Par exemple, si un lot de 10 stylos à 100 F a un pourcentage de réduction de 20 %, le prix du lot sera de 10 x 100 x (100 - 20) / 100 = 800 F.

Proposer une implémentation en utilisant un design pattern vu en cours.
Faîtes préalablement le diagramme de classes UML correspondant.
Identifier les « rôles » de chaque classe dans le design pattern

Question #9.
Dans un second temps, on suppose que les lots comportent plusieurs types d'articles.
Proposer une implémentation.

Exercice 3. En observation

Question #10 : Implémenter en Swing/AWT une application graphique très simple avec un bouton et à chaque fois qu'un utilisateur clique sur le bouton deux messages distincts apparaissent (les deux messages seront affichés dans deux méthodes différentes).
Vous utiliserez un design pattern vu en cours.
Identifier les « rôles » de chaque classe dans le design pattern.

Exercice 4. Tueur de dragons

On souhaite implémenter un jeu vidéo dans lequel un dragon doit éliminer des adversaires.
En cours de partie, un dragon peut utiliser différentes techniques : combat corps à corps, cracher du feu, ou lancer des projectiles.
(L'objectif n'est pas de construire une interface graphique complète, mais simplement d'implémenter les classes Java nécessaires)

Question #11 : Implémenter une classe Dragon capable d'utiliser ces différentes techniques en cours de jeu. (On se contentera d'afficher une chaîne de caractère pour montrer que la technique est effectivement utilisée en cours de jeu)
Vous utiliserez un design pattern vu en cours.
Identifier les « rôles » de chaque classe dans le design pattern.
Question #12 : Une partie possède un niveau de difficulté choisi lors de la création de la partie et pouvant changer au cours de celle-ci. Il existe 3 niveaux de difficulté : facile, normal et difficile. Rajouter cette fonctionnalité à l'aide d'un design pattern.

Question #13 (option) : En fonction du niveau de difficulté, un dragon utilisera à plus ou moins bon escient ses aptitudes… Dans le niveau « facile », il choisira automatiquement la meilleure technique d'exécution ; dans le niveau « normal », l'utilisateur doit explicitement choisir la technique d'exécution (via une combinaison de touches) ; dans le niveau « difficile » le choix de la technique d'exécution est sélectionné aléatoirement (l'utilisateur peut quand même re-prendre le contrôle).
Implémenter une telle solution.

Exercice 5. Télécommande

Dans le dossier « DP », il y a un dossier « telecommand » qui contient l'implémentation d'un programme Java.

Question #14. Que fait ce programme ? Quel design pattern est-utilisé ?
Faîtes un diagramme de classes UML de l'application. Identifier les « rôles » de chaque classe dans le design pattern.

Question #15. Etendre le programme précédent pour que la télécommande puisse être invoqué avec des numéros (0, 1, .., 9). Modifier RemoteControl et inventer de nouvelles fonctionnalités (e.g., éteindre la cuisine, la TV, régler la luminosité de la pièce principale, etc.) en utilisant le même design pattern.

Question #16. Ajouter la possibilité d'annuler une commande (par exemple, si j'ai éteint la cuisine et que j'annule la commande, je dois retrouver la même luminosité dans la cuisine)