Cours Modélisation et Conception Orientée Objet 2009-2010
TD n° 6
Design Pattern Composite

1/ On modélise la grammaire abstraite d'un langage d'expressions arithmétiques simples. Ce langage offre des constantes (3, 5,6 ...) des variables (a, b, ...) et des opérateurs unaires (!,-), binaires (+,* ...) et ternaire (_ ? _ : _).

(3 + 2x)*(7/y -2) est une expression acceptable.

Définir une hiérarchie de classes organisées selon le design pattern "composite" qui implémente ce langage abstrait.
Observer que ces classes constituent décrivent les données qui seraient construites par un compilateur pour ces expressions.
Observer également que la syntaxe concrète du langage peut changer (i.e. de forme infixe, préfixe (polonaise) ou postfixe (polonaise inverse) : ici par exemple ((3 (2 x *) +) ((7 y /) 2 -) *)

Implémenter ces classes en Java.
Implémenter une fonction d'évaluation qui s'appuie sur un dictionnaire (HashMap par exemple) donnant des valeurs aux variables.

2/ Définir une hiérarchie de classes d'interfaces graphiques organisées selon le design pattern composite. On distingue entre les objets "terminaux" : bouton, label, boite à cocher etc, et les objets non terminaux : 'panel' (le container générique), menubar, menu, list etc...

On intégrera dans la hiérarchie des types de panneaux qui structurent leur contenu : par exemple SplitPane (un panneau qui combine deux objets séparés par une barre mobile horizontale où verticale), ou TabbedPane (un panneau qui groupe des objets organisés par onglets).

Compléter par la définition d'une fonction d'affichage (simulée) récursive.

Cours Modélisation et Conception Orientée Objet
TD n° 7
Design Pattern Iterator

1/ Concevoir une hiérarchie de classes d'itérateurs pour votre hiérarchie de classes de containers. Veiller à ce que chaque classe de container implante les itérateurs qui lui sont spécifiques.

2/ Veiller à ce que les fonctions de recherche ou de parcours mises en oeuvre par ces classes soient réalisées ou proposées exclusivement via ces itérateurs.

3/ Compléter cet existant par une hiérarchie de classes d'itérateurs offrant des services avancés:
- parcours arrière
- réinitialisation
- suppression/insertion

4/ Observer que les fonctions d'insertion doivent obligatoirement être mises en oeuvre par les itérateurs dans le cas des collections ordonnées. Compléter votre api pour le permettre, sans pour autant supprimer les fonctions d'insertion habituelles de votre classe de départ.

5/ Observer que les fonctions de suppression de l'objet courant peuvent être implantées de manière plus efficace par les itérateurs. Compléter votre api pour le permettre, sans pour autant supprimer les fonctions d'insertion habituelles de votre classe de départ.

6/ (subsidiaire) Observer que les fonctions d'insertion suppression posent des problèmes avec les threads. Etudier comment contourner ce problème en faisant en sorte que ces itérateurs acuièrent des "locks" sur les objets qu'ils parcourent.

Cours Modélisation et Conception Orientée Objet
TD n° 8
Design Patterns Command, Adapter, Singleton

1/ (Modélisation) Compléter la hiérarchie de classes d'interface fictives de la séance 6 de manière à ce qu'elles permettent l'enregistrement de callbacks selon le design pattern "Command".

2/ On veut exécuter du code au début et à la fin de toutes les fonctions d'une classe. Compléter une de vos classes container de manière à ce que ce mécanisme, mis en oeuvre selon "Command" permette au choix d'afficher une simple trace d'exécution, de construire une pile d'appels, ou de 'profiler' le programme (compter le nombre d'appels, et les temps passés): traiter le premier et troisième cas

3/ Permettre que les callbacks puissent être ajoutés de manière à former une liste chaînée.

4/ Faire en sorte que la valeur de retour de la fonction appelée pour un callback permette de bloquer l'exécution des callbacks suivants dans la liste

5/ Tester la réalisation du pattern "Adapter" sur une classe de votre choix

6/ Réaliser une classe gérant son unique instance selon le pattern "Singleton"