PARIS DESCARTES
IUT

DUT Informatique - 2ème année

CPOA – Conception et Programmation à Objets Avancées

TRAVAUX PRATIQUE – Semaine n°2

Thèmes

- Dépendances, contraintes de couplage, (refactoring)

Note : vous devez pouvoir faire le TP sans outil. Puck n'est qu'une aide accessoire qui ne sera pas disponible au DST.

1. Point de départ et objectifs de conception

Créez un projet java *matrices* et importez les sources du répertoire `énoncé` sur le serveur commun.

Le projet est un embryon d'un programme de calcul matriciel. Rappelons qu'une matrice HxL est un objet mathématique similaire à un tableau à deux dimensions : sa hauteur H et sa largeur L. Si M est une matrice 2x3 alors ses postes sont

M11 M12 M13

M21 M22 M23

Un moyen simple d'implémenter une matrice est donc tout simplement à l'aide d'un tableau à deux dimensions (voir la classe **`MatricePleine`**). Toutefois, si de nombreux postes de la matrice sont nuls alors il est possible de gagner de la place en ne stockant que les coordonnées et la valeur des postes non nuls. (Voir la classe **`MatriceCreuse`**). Par exemple si seul le poste d'indices (1,3) est non nul alors une matrice creuse associera simplement à la clé (1,3) la valeur de ce poste (mettons 5) ce qui est plus compacte que de stocker le tableau complet

0 0 5

0 0 0

La classe **`Calcul`** doit permettre de manipuler des matrices (une seule dans cette version) mais sans avoir à se préoccuper de son implémentation, c'est-à-dire **sans dépendre de la classe utilisée pour représenter les matrices** (`MatriceCreuse`, `MatricePleine` voire d'autres classes à l'avenir).

2. Puck

Puck est présenté dans le cours. Vous pouvez ou non l'utiliser pour le TP et pour les suivants soit juste pour montrer les dépendances soit, si vous précisez une contrainte de couplage, pour montrer les dépendances interdites.

3. Architecture

Dessinez un extrait de l'architecture actuelle en indiquant les paquetages, les classes et leurs relations (sous-typage et dépendances) mais sans préciser les méthodes ni les attributs. N'indiquez que les classes publiques et n'indiquez pas les classes de test.

Il s'agit un résumé du graphe de dépendances dans lequel on marque seulement les dépendances entre classes et interfaces mais pas chaque dépendance vers les méthodes ou attributs sauf lorsqu'elles sont interdites.

4. Contrainte de couplage

1. Écrivez en langage naturel une contrainte de couplage qui traduise les objectifs de conception explicités dans le paragraphe 1 : il faut cacher quoi de quoi ?
2. Renommez le fichier `template.pl` en `decouple.pl` puis écrivez-y cette contrainte sous une forme compréhensible par l'outil Puck.
3. (facultatif) A l'aide de Puck affichez le graphe de dépendances qui prend en compte la contrainte de couplage. Si vous n'utilisez pas Puck indiquez les dépendances interdites sur le diagramme d'architecture.

5. Tests

Avant de procéder au refactoring du code, écrivez deux classes de test, MatricesTest et CalculTest en utilisant Junit. Ces tests vous permettrons de vérifier que le code fonctionne toujours après le refactoring (on ne refactore jamais sans filet).

6. Refactoring

Refactorez le programme pour ôter les dépendances nuisibles. Raisonnez sur le graphe et procédez étape par étape en ôtant une dépendance ou un groupe de dépendances à la fois.

Lancez régulièrement Puck pour vérifier les dépendances nuisibles qui restent à ôter. A la fin du refactoring **redessinez l'architecture** que vous avez obtenue.

N'oubliez-pas qu'il peut être nécessaire de modifier decouple.pl si vous déplacez des classes par exemple dans d'autres paquetages ou si vous ajoutez des classes.

Une première étape conseillée consiste à déplacer les classes qui ne doivent pas être contenues (leur nom est souligné et en rouge) dans leur paquetage actuel vers un autre paquetage que vous créerez pour l'occasion.

Une deuxième étape consiste à introduire une interface pour masquer les classes concrètes à cacher. Choisissez bien le paquetage où placer cette interface.

Enfin si le temps le permet ôtez les dernières dépendances qui sont sans doute des créations d'instance, par exemple en introduisant une factory comme vu en première année.

Il n'est pas prioritaire de traiter les dépendances des classes de test mais vous pouvez le faire s'il vous reste du temps.