PARIS DESCARTES
IUT

DUT Informatique - 2ème année

CPOA – Conception et Programmation à Objets Avancées

TRAVAUX PRATIQUE – Semaine n°4

Thèmes

- Interface Segregation Principle
- Refactoring pour ôter des dépendances nuisibles

Exercice 1 Interface Segregation Principle

Importez le code de l'exo 1 du TP4 sur le serveur commun. **Il est inutile d'utiliser Puck pour cet exercice.**
Le problème ici concerne les classes `Client` et `Fournisseur` qui n'ont pas besoin de certaines méthodes de l'interface `IContact`.
Le risque est de débiter par erreur un fournisseur ou de payer par erreur un client. Il faut aussi éviter les méthodes qui retournent des pointeurs nuls.
Remaniez le code pour que les classes `Client` et `Fournisseur` n'implémentent que les méthodes nécessaires. La classe Vente ne doit pas dépendre de classes concrètes.
Pour cet exercice, vous pouvez modifier (aussi peu que possible) les tests et la partie publique des classes et interfaces.

Question subsidiaire : comment faire pour qu'un client soit aussi fournisseur ?

Exercice 2 Refactoring pour ôter les dépendances nuisibles

Importez le code de l'exo 2 du TP4 sur le serveur commun. Ce code est un embryon de système de fichiers. Pour le moment il y a deux sortes de fichiers, les répertoires et les fichiers simples, mais il doit être possible d'en ajouter facilement. Ceci implique que les clients du système de fichiers ne doivent pas être impactés par un tel ajout. De plus, comme le système de fichier lui-même va sans doute devenir complexe il faudrait qu'il ne soit impacté qu'au minimum par un tel ajout.

Écrivez en langage naturel une contrainte de couplage qui traduise les objectifs de conception explicités dans le paragraphe précédent : il faut cacher quoi de quoi ?

Pour utiliser Puck écrivez une contrainte decouple.wld.

Avant de remanier le code, commencez par ajouter des tests unitaires pour File et Directory (on ne remanie pas sans filet).

La suppression des dépendances sur les créations d'instances : new File() et new Directory() **ne sera pas faite**.

En créant une interface IFile vous devez arbitrer le principe ISP différemment de l'exercice 1 ici. En effet, ici la priorité est que l'on puisse traiter aussi uniformément que possible fichiers et répertoires. Il faut donc faire l'union des

méthodes des classes correspondantes quitte à vérifier avec une assertion qu'on n'ajoute rien à un fichier et qu'on ne cherche pas dans un fichier.

Ajoutez deux tests dans la classe FileTest pour vérifier que des assertions empêchent bien les opérations impossibles.