Decorator

Exercice 1 (L'entreprise IT)

Le personnel d'une entreprise IT (disons, de développement d'applications) occupe des postes différents, comme celui de chef de projet, d'architecte logiciel, d'analyste (fonctionnel) et de développeur (appelé aussi "analyste programmeur" car ça fait plus classe). En gros :

- le chef de projet gère et coordonne une équipe en s'assurant du bon avancement du projet,
- l'architecte définit les outils qui seront utilisés pour ce projet (ex : pour ce projet, on va utiliser C++ avec la bibliothèque Boost),
- l'analyste fonctionnel va définir les fonctionnalités du logiciel à développer.
- le développeur se tait et programme.

Cependant dans les petites structures, une même personne se retrouve bien souvent avec plusieurs de ces casquettes. Nous allons donc implémenter cela en Java avec le pattern Decorator.

Notre programme devra contenir les classes et interfaces suivantes :
- une classe Projet, avec une description et une méthode affichant la description,
- un décorateur,
- ChefDeProjet qui aura une liste de membre d'équipe (et les méthodes pour ajouter/supprimer quelqu'un), plus une méthode superviser() modifiant la description du projet (par exemple, un ajoutant "supervisé par NomDuChef"),
- Architecte avec une méthode definirOutils() modifiant la description du projet,
- Analyste avec une méthode definirFonctions() modifiant la description du projet,
- Developpeur avec une méthode coder() modifiant la description du projet.
- une classe abstraite Employé, avec un nom et le projet sur lequel il travaille (et assesseurs, ...), et surtout la déclaration des méthodes de tout ce que peut faire un chef de projet, un architecte, ...
- une seule classe EmployéConcret (avec constructeur, ...),

1. Faire le diagramme UML de ce programme.
2. Écrire les classes et interfaces en Java en prenant ceci en compte : on peut facilement imaginer qu'un employé ait un salaire de base, et que chaque fonction apporte un complément de salaire plus ou moins important. Après avoir modifier le diagramme UML, ajouter le calcul et l'affiche du salaire dans le programme.

Exercice 2 (Encapsuler un entier)

On va refaire la classe Integer, en partie. L'objectif va être de définir une classe Entier contenant un entier (normal, en décimal) et ses décorateurs pour convertir l'entier en binaire, octal et hexadécimal, avec un décorateur par base.

Pour cela, la classe Entier et ses décorateur contiendront une méthode `base()` affichant à l'écran la valeur de l'entier dans la base adéquate. Pour les décorateurs, l'appel à `base()` devra également appeler `base()` de l'objet décoré.

Exercice 3 (Le concessionnaire automobile)

Vous dirigez une concession BMW (tant qu'à faire), et vous voulez proposer à vous client des 325, 330, et 535, essence ou diesel, avec un choix de différentes options (couleur métallisée, GPS, intérieur cuir, ...), tout ceci ayant bien sûr un impact sur le prix. Même topo : diagramme UML et code Java.

Exercice 4 (Décorer InputStream)

Faire une classe ReverseInputStream décorant InputStream (conseil : dérivez de FilterInputStream) et proposant une méthode retournant la stream inversée octet par octet.