Université Abou Beckr Belkaid
Master 2 Génie Logiciel
Année Universitaire 2014/2015

Correction de l'Examen Final

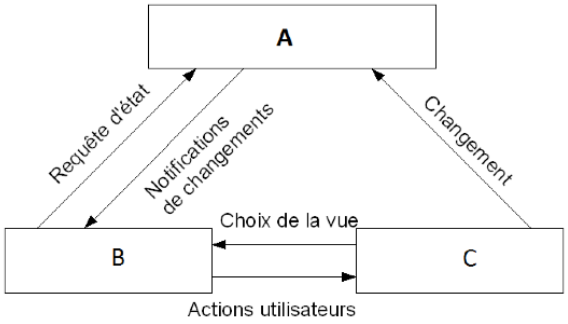
—

Architecture et Développement Logiciels

—

Partie A (8 points) : Cochez la (ou les) bonne(s) réponse(s).

Questions	Réponses
1. Un design pattern est :	■ une norme de description des interfaces entre les composants d'une architecture logicielle orientée objet.
	■ une définition des principes de conception.
	☐ une définition des implémentations spécifiques à des principes de conceptions
	☐ aucune réponse juste.
2. Le design pattern Factory est :	■ un patron de création.
	☐ un créateur d'objet singleton.
	■ un créateur d'objets tous décrit par la même interface.
	☐ aucune bonne réponse.
3. Le design pattern Adaptateur :	☐ est un patron dans lequel l'adaptateur et l'adapté implémente la même interface.
	■ correspond à une classe qui sert d'intermédiaire entre un appelant et un appelé qui sont incompatibles entre eux
	☐ appartient aux patrons de comportement.
	☐ aucune bonne réponse.

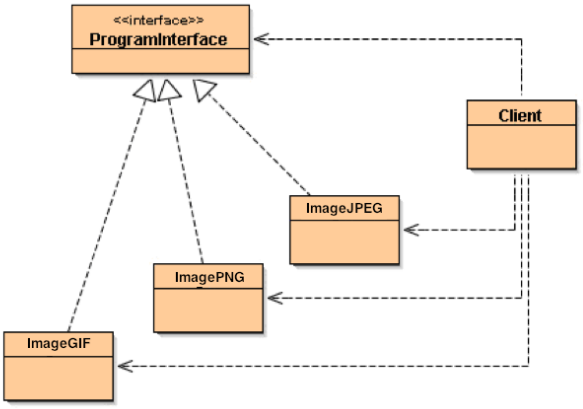
Questions	Réponses
La figure suivante représente le pattern Modèle-Vue-Contrôleur :  	

Questions	Réponses
1. Les lettres A, B et C sont définit comme suite :	■ A = Modèle, B = Vue, C= Contrôleur. ☐ A = Contrôleur, B = Vue, C= Modèle . ☐ aucune bonne réponse.
2. Le pattern MVC est utilisé pour :	☐ séparer le code technique du code métier. ■ mettre en avant l'interface utilisateur et la rendre indépendante des couches plus basses du modèle. ■ la réalisation d'interface homme-machine ☐ aucune bonne réponse.
3. Les EJB (Entreprise Java Bean) :	■ permettent de construire des applications distribuées. ■ définissent un standard JavaBean pour faciliter la réutilisation et l'interopérabilité des composants middleware. ■ définissent l'un des modèles de composants principaux de J2EE ☐ aucune bonne réponse.
4. Un EJB session est :	☐ un bean exécuté du côté client. ■ la partie de l'application qui prend en charge la logique métier. ☐ composé obligatoirement de deux interfaces local et distante. ☐ aucune bonne réponse.
5. La programmation orientée aspect :	☐ est un modèle de programmation. ■ permet la mise en œuvre de la séparation des préoccupations. ☐ remplace la programmation orientée objet, en corrigeant ses limitations. ☐ aucune bonne réponse.

Partie B (12 points) : Questions libres.

Questions	Réponses

Dans une application de création de photos numériques, un client à la possibilité de créer des images matricielles (bitmap) de trois formats : GIF, JPEG, PNG.
Dans une première implémentation, le client définit le format de l'image à la création et cela dans son propre programme principale. Voici le diagramme de classe correspondant :



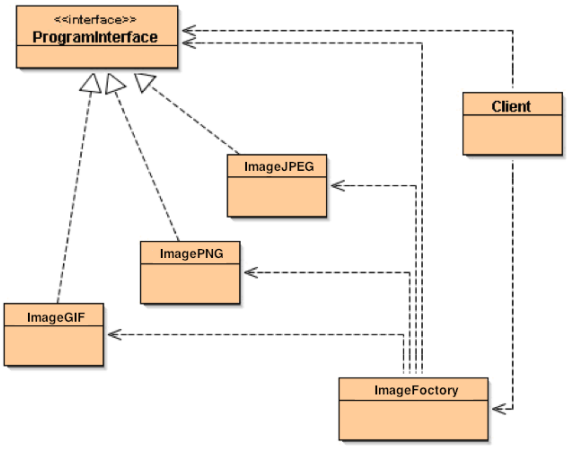
Le problème de cette conception est que les images sont construites dans la fonction main des clients. Si l'application est mise à jour et que la façon dont les images sont générées change, cela revient à changer le code de tout les clients. Ce qui est intolérable dans une application flexible et évolutive.

Questions :

1 quel design pattern proposez vous pour éviter ce problème ? justifiez votre réponse.(3 points)
2 Donner la nouvelle version du diagramme de classe de votre application.(2 points)
3 Pouvez-vous ajouter le format TIFF aux types d'images générées ? Si oui, comment procéderiez vous (en justifiant) ? (1 point)

Réponses :

1 Le design pattern adéquat à ce problème est le design pattern **Factory**. En effet, s'il y avait plusieurs classes Client, chacune aurait une fonction de création des objets image matricielle, et si l'on voulait modifier la façon dont les images sont générés, il faudrait modifier chacune des fonctions de création des images dans les classes Client. Pour éviter ça, on va déléguer la création des objets de type image à une classe dont le nom sera **ImageFactory**. Les clients devront construire une instance de cette classe afin de pouvoir y utiliser la fonction de création des images qui y sera stockée.
2 Voici le diagramme de classe que vous devriez obtenir :



Questions	Réponses
3 Le pattern **Factory** permet au client de ne pas avoir connaissance du produit (GIF, JPEG ou PNG), et de déléguer les détails de la production à une autre classe. Si on souhaite rajouter un produit (une image au format TIFF), on doit juste éditer la factory. Grâce à la factory, on évite à la classe Client d'avoir connaissance des différentes instances de **ProgramInterface**.	

Questions	Réponses

Dans cette exercice nous supposons l'existence d'un système de transaction bancaire implémenté dans un langage orienté objet. La partie implémentation orientée aspect du système est donnée comme suite :

```
public aspect AspectDemo {
    Log log = new Log("fichier");
    // Commentaire 1
    pointcut appelTransaction(Information info):
        call(void ProcessusTransaction.effectuerTransaction(Information)) && args(info);
    // Commentaire 2
    before(Information info): appelTransaction (info) {
        log.enregistrer("Tentative transaction:" + info);
    }
}
```

Questions :

1 Remplacer les commentaires 1 et 2 en expliquant en **détail** l'instruction qui suit chaque commentaire.(4 points)
2 Ajouter à l'aspect **AspectDemo** les instructions adéquates afin de remplacer l'appel à la méthode ***void ProcessusTransaction.effectuerTransaction(*)*** par le message d'erreur "transaction impossible" et cela en l'affichant dans la console et aussi en l'enregistrant dans le fichier *log*.(2 points)

Réponses :

1 L'aspect **AspectDemo** assure que chaque fois que la méthode ***effectuerTransaction*** de la classe ***ProcessusTransaction*** est appelée, on enregistre les informations passées en argument.

- Commentaire 1 : le pointcut qui à le nom ***appelTransaction*** sélectionne le JoinPoint correspondant à l'appel de la méthode **effectuerTransaction** de la classe ***ProcessusTransaction*** et qui prend un objet de type ***Information*** en paramètre. Ce paramètre est récupéré par le pointcut **args** afin qu'il puisse être utilisé comme paramètre du pointcut ***appelTransaction***.
- Commentaire 2 : C'est l'advice qui avant le pointcut ***appelTransaction*** enregistre dans le fichier **log** le message " Tentative transaction" en plus de l'objet du type ***Information*** qui est récupéré du contexte de l'appel du pointcut.

2 La partie à ajouter à l'aspect **AspectDemo** :

```
pointcut remplaceTransaction():
    call(void ProcessusTransaction.effectuerTransaction(*));

void around(): remplaceTransaction() {
    System.out.println("Tentative impossible!");
    log.enregistrer("Tentative impossible!");
}
```

Bon courage et bonne continuation.