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

Examen Final

—

Architecture et Développement Logiciels

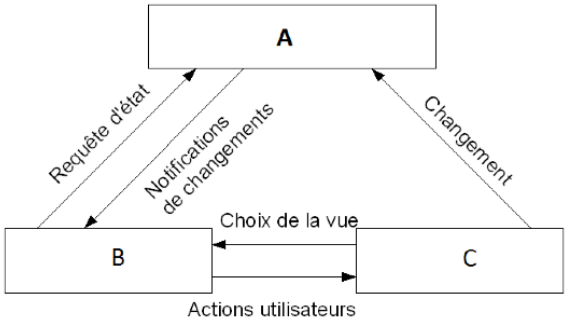
—

Nom	
Prénom	
Numéro Etudiant	

Remarques :
- Les documents ne sont pas autorisés ainsi que les appareils électroniques (PC, Tablette, téléphone, etc).
- La première partie A est sous forme de QCM à choix multiples.
- La seconde partie B est sous forme de questions libres à répondre sur la double feuille.

Partie A : 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.

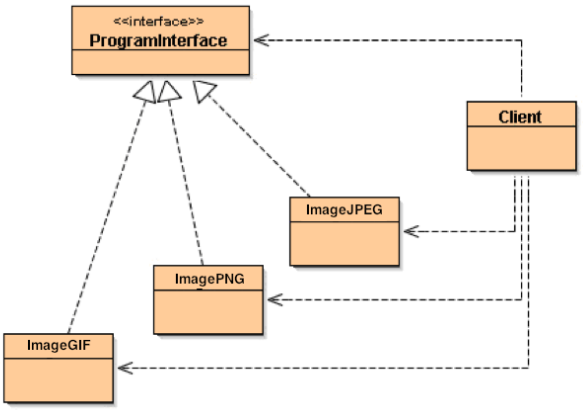
Questions	Réponses
La figure suivante représente le pattern Modèle-Vue-Contrôleur :  	

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 : 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.
2 Donner la nouvelle version du diagramme de classe de votre application.
3 Pouvez-vous ajouter le format TIFF aux types d'images générées ? Si oui, comment procéderiez vous (en justifiant) ?

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.
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*.

Bon courage et bonne continuation.