UE Ingénierie Logicielle - Design Patterns
Contrôle Terminal - Session 1

Durée : 1h30
Aucun document n'est autorisé *à l'exception du support de cours qui est fourni avec le sujet.*
Le barème est approximatif. Il est donné à titre indicatif.

Les réponses doivent être claires, concises, précises et présentées lisiblement.
Il est inutile de recopier le support de cours*. Au contraire, les recopies seront évaluées négativement.*

EXERCICE 1 [*environ 10 points*]

Les questions sont indépendantes.

1. [*4 points*] Indiquer si les affirmations de l'Annexe 1 sont vraies ou fausses.

2. [*3 points*] Mme Cérès, agricultrice 4.0, a équipé un de ses champs :

   - d'une sonde qui mesure des valeurs environnementales (humidité du sol, température, degré d'hygrométrie…) et produit des mesures de type `Mesure`,
   - de différents appareils (arroseurs automatiques, drones autonomes pulvérisateurs, robots de désherbage…) connectés à la sonde qui lui permettent d'entretenir automatiquement ses cultures en fonction des mesures.

   L'ensemble des appareils « connectés » à la sonde peut évoluer dans le temps : un appareil peut être ajouté ou être retiré au cours de l'exécution. D'autre part, quand la mesure change, les appareils connectés à la sonde doivent recevoir la nouvelle valeur : pour cela, leur interface offre la méthode `void recevoir(Mesure mesure)`.

   Quel design pattern du GoF permet de mettre en œuvre la proposition ci-dessus ? Justifier. Préciser les participants et leurs rôles.
   NB : La conception détaillée (diagramme de classes) de la solution n'est pas demandée.

3. [1 *point*] « Fabrique simple » n'est pas un design pattern du GoF mais une bonne pratique de conception[1] dont l'objectif est de séparer les responsabilités et de réduire le couplage.
   Quel est le principe de la solution ?

4. [*2 points*] Dans une application structurée conformément au design pattern Fabrique Abstraite, que doit-on faire pour intégrer une nouvelle famille de produits et sa fabrique ? Quel est l'impact de cette opération sur le code existant ?

[1] Ne perdez pas du temps à chercher Fabrique Simple dans le support de cours, vous ne l'y trouverez pas.

EXERCICE 2 [*environ 4 à 5 points*]

On considère une application de diffusion de documents : photos, podcast audios, vidéos… Parmi les vidéos, il y a des vidéos promotionnelles qui peuvent être des témoignages ou des publicités. Cette application est structurée conformément au diagramme de classes en Annexe 2.

Une évolution est demandée concernant les vidéos : pouvoir leur ajouter diverses « fonctionnalités », par exemple un sous-titrage, un lien cliquable vers la documentation d'un produit, un widget interactif, etc. Pour mettre en œuvre la solution, l'équipe de développement a choisi le design pattern Décorateur.

1. Mettre en œuvre la solution au moyen du design pattern Décorateur : compléter le diagramme de classes de l'Annexe 2 et donner les quelques lignes de code Java qui précisent la solution.

2. Donner les quelques lignes de code Java qui réalisent la création d'un document de type Vidéo puis sa « décoration »

EXERCICE 3 [*environ 5 à 6 points*]

On considère une application dans laquelle un demandeur (instance de la classe `Requester`) peut demander à un fournisseur (de type `Provider`) la réalisation d'un service en indiquant son nom (de type `Name`). Pour cela, le fournisseur offre la méthode `serve()`. Le fournisseur doit d'abord « résoudre le nom » c'est-à-dire trouver l'URL du service à partir du nom, au moyen de la méthode « protected » `solve()` implémentée dans la classe Provider.

Cette application est structurée conformément au diagramme de classes fourni en Annexe 3. Les classes `P1`, `P2`, `P3`… représentent différents types de fournisseurs de service. Elles implémentent chacun la méthode `serve()` à leur façon en utilisant la méthode `solve()` définie dans la classe Provider.

Dans le cadre d'une demande d'évolution, il faut intégrer à l'application un nouveau type de fournisseur de service (classe NSP). Pour demander un service à un fournisseur de type NSP, il faut lui passer directement l'URL du service en paramètre.
Pour réaliser cette évolution, toute modification du code existant est interdite, que ce soit le code de l'application à maintenir ou la classe NSP.

1. Quel est le problème ?

2. Quel design pattern du GoF permet de répondre à ce problème ? Justifier votre réponse, et préciser les rôles des participants.

3. Donner le diagramme de classes qui réalise la solution proposée conformément au pattern choisi : compléter le diagramme de classes de l'Annexe 3 et donner les quelques lignes de code Java qui précisent la solution.

4. Donner les quelques lignes de code Java qui réalisent la création d'un demandeur et sa configuration avec un fournisseur de type NSP, puis l'exécution de la demande de service.

ANNEXE 1 (exercice 1.1) **NOM :** **PRENOM :**

Pour chaque affirmation ci-dessous, **entourer la bonne réponse**.
Rendre la feuille avec la copie. Ne pas oublier d'indiquer votre nom et prénom.

Notation : par affirmation, +0,33 pour une réponse correcte, -0,33 pour une réponse incorrecte, 0 en l'absence de réponse.

1. Le design pattern Patron de Méthode vise l'organisation du code.
   VRAI --- FAUX

2. Quand on met en œuvre le design pattern Patron de Méthode, la classe mère de la hiérarchie est obligatoirement une classe abstraite.
   VRAI --- FAUX

3. La principale différence entre les patterns Patron de Méthode et Stratégie réside dans l'utilisation de l'héritage pour Patron de Méthode et de l'association pour Stratégie.
   VRAI --- FAUX

4. Le design pattern Patron de Méthode peut être combiné avec le design pattern Stratégie pour réaliser l'implantation des stratégies concrètes.
   VRAI --- FAUX

5. Quand on met en œuvre le pattern Stratégie, il n'est pas obligatoire de définir dans la classe Utilisateur la méthode publique setStratégie() qui permet de donner une stratégie à l'objet utilisateur.
   VRAI --- FAUX

6. Pour mettre en œuvre un objet dont le comportement dépend d'un état, on doit obligatoirement utiliser le pattern State (Etat).
   VRAI --- FAUX

7. Pour mettre en œuvre le pattern Adaptateur, il faut modifier la classe Client ou la classe Adaptée.
   VRAI --- FAUX

8. Dans la mise en œuvre du pattern Adaptateur, le supertype Cible (qui définit le besoin du client) est obligatoirement une interface (au sens « Java »)
   VRAI --- FAUX

9. Une façade est un adaptateur qui cache au client plusieurs objets adaptés.
   VRAI --- FAUX

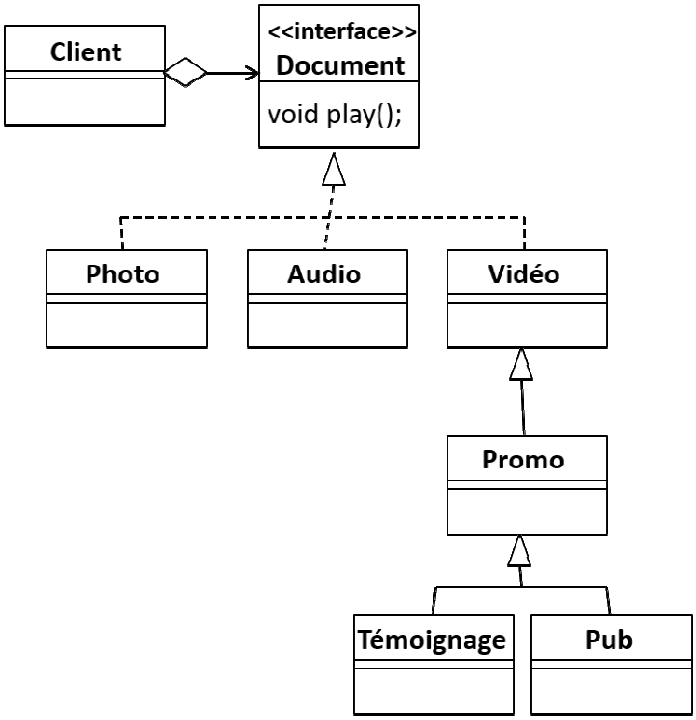
10. Une façade peut offrir des services de plus haut niveau que les services offerts par les objets du sous-système.
    VRAI --- FAUX

11. En mode « pull », un observateur reçoit une notification de changement d'état du sujet et doit faire une demande au sujet s'il veut connaitre le nouvel état.
    VRAI --- FAUX

12. Dans la mise en œuvre du pattern Observateur, la définition de l'interface Observateur dépend du mode de transfert de l'état (« pull » ou « push »).
    VRAI --- FAUX

ANNEXE 2 (exercice 2) **NOM :** **PRENOM :**

Rendre la feuille avec la copie. Ne pas oublier d'indiquer votre nom et prénom.



ANNEXE 3 (exercice 3) **NOM :** **PRENOM :**

Rendre la feuille avec la copie. Ne pas oublier d'indiquer votre nom et prénom.