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 5 points)*

Les questions sont indépendantes.
Pour les réponses, le nombre de lignes à produire est approximatif. Il est donné à titre indicatif.

1. Quel est le nom du pattern dont on peut dire qu'il permet d'envelopper un ensemble d'objets pour en faciliter l'accès ? (donnez seulement le nom sans explication)

2. Quelle est la principale différence entre les patterns State (Etat) et Stratégie ? (2 lignes)

3. Quand on met en œuvre le pattern Stratégie, il n'est pas obligatoire de définir la méthode `void setStratégie(Stratégie s)` dans la classe Utilisateur.
   Quand peut-on ne pas définir cette méthode ? Quelle est alors la solution pour associer un objet stratégie à un objet utilisateur ? (2 à 4 lignes)

4. Dans la solution du pattern Proxy, pourquoi la classe `Proxy` doit-elle implanter la même interface que la classe `SujetRéel` ? (2 à 3 lignes)

EXERCICE 2 *(environ 8 points)*

On considère l'application dont le code est fourni en annexe, dans laquelle un écran affiche la vitesse instantanée d'un véhicule. Dans cette application, la mesure de la vitesse en kilomètres par heure (Km/h) est fournie sur demande par un capteur associé à l'écran. Il existe deux types de capteurs, « Chacal » et « TimTim ».

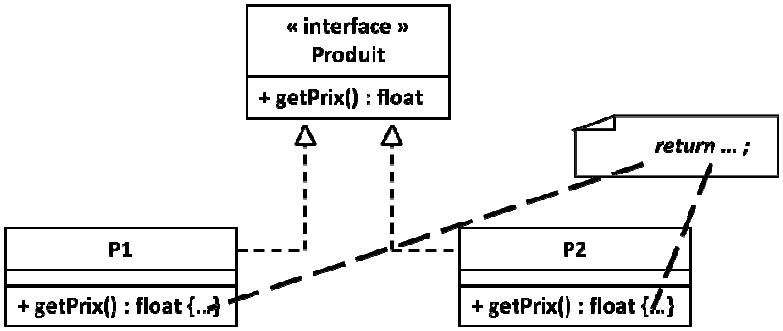
Dans le cadre d'une demande d'évolution, il faut intégrer à l'application un nouveau type de capteur « Ways » qui mesure la vitesse en Miles per hour[1] (Mph). L'implantation de ce type de capteur est définie par la classe `Ways` (également fournie en annexe). Pour réaliser cette évolution, toute modification du code existant est interdite, que ce soit le code de l'application à maintenir ou la classe `Ways`.

1. Quel design pattern du GoF permet de répondre à ce problème ? Justifier votre réponse.
2. Donner le diagramme de classes qui réalise la solution proposée, ainsi que les quelques lignes utiles de code Java (les « bouts de code » qui précisent la solution).
3. Donner les quelques lignes de code Java qui complètent la classe `Test` et qui réalisent l'association d'un capteur de type « Ways » à un écran puis une demande d'affichage de la vitesse instantanée à cet écran.

[1] Pour convertir une vitesse en Mph en une vitesse en Km/h, il faut multiplier la vitesse en Mph par 1,61.

EXERCICE 3 *(environ 7 points)*

On considère une application qui permet de définir des produits avec un prix. Ces produits peuvent retourner leur prix via la méthode `getPrix()`. Concrètement, on suppose qu'il existe deux types de produits, `P1` et `P2`. Cette application est structurée conformément au diagramme de classes ci-dessous.



On veut pouvoir ajouter des frais ou des taxes au prix de base d'un produit, ou diminuer ce prix par des réductions. Le prix d'un produit, retourné par la méthode `getPrix()`, sera ainsi calculé à partir du prix de base et par application éventuelle des divers frais et/ou de taxes et/ou de réductions…

Le problème est d'étendre l'application sans modifier l'existant (c.-à-d. sans modifier ni `Produit`, ni `P1`, ni `P2`, voir le diagramme ci-dessus) de telle sorte que l'on puisse remplacer un produit par un autre produit avec des frais et/ou des taxes et/ou des réductions... Ce remplacement pourra même être fait dynamiquement. L'application obtenue permettra la manipulation de produits, quelque soit le type de base, avec ou sans frais, taxés ou pas, avec réduction ou pas, etc. Elle devra être facilement extensible dans le futur (sans modification de l'existant) en cas de nouveau frais ou de nouvelle taxe ou de nouvelle réduction.

Concrètement, notre application doit permettre entre autres :

- d'appliquer au produit une réduction de 10%,
- d'ajouter des frais de port fixes (d'une valeur fixée arbitrairement à 150 euros),
- d'ajouter une taxe à l'exportation de 20%.

1. Quel design pattern du GoF faut-il utiliser ici ? Justifier.
2. Donner le diagramme de classes qui met en œuvre la solution conformément à ce pattern, et préciser qui sont les participants. Donner le code Java de la méthode `getPrix()` dans les classes ajoutées.
3. Donner le programme Java qui crée un produit de type `P2`, lui ajoute les frais de port, puis la taxe à l'exportation (la taxe s'appliquant aussi aux frais de port) et enfin la réduction de 10% sur le tout.

Annexe (exercice 2)

```
public interface Capteur {
      public double donnerVitesse();
}
public class Chacal implements Capteur {
      public double donnerVitesse() {
            double vitesse = ……. // mesure de la vitesse en Kmh
            return vitesse;
      }
}
public class TimTim implements Capteur {
      public double donnerVitesse() {
            double vitesse = ……. // mesure de la vitesse en Kmh
            return vitesse;
      }
}
public class Ecran {
      Capteur monCapteur;
      public Ecran(Capteur monCapteur) {
            this.monCapteur = monCapteur;
      }
      public void afficherVitesse() {
            System.out.println("Vitesse  =  "+monCapteur.donnerVitesse()+"
Kmh");
      }
}
public class Test {
      public static void main(String[] args) {

            Chacal chacal = new Chacal();
            Ecran écran1 = new Ecran(chacal);
            écran1.afficherVitesse();

            TimTim timtim = new TimTim();
            écran2 = new Ecran(timtim);
            écran2.afficherVitesse();
      }
}
```

```
public class Ways {
      public double getSpeed() {
            double speed = ……. // mesure de la vitesse en Mph
            return speed;
      }
}
```