UE Ingénierie Logicielle - Design Patterns – Feuille d'exercices n°1

EXERCICE 1

1) Pour le code ci-dessous, donner le diagramme de classes et mettre en commentaire les instructions qui provoquent une erreur.

```
class A {
        private B myB;
        void setMyB(B myB) {this.myB = myB;};
}
class B {
}
interface C {
}
class D {
}
class E extends B {
        private D myD;
        void setMyD(D myD) {this.myD = myD;};
}
class F extends B implements C {
}
class G {
        private F myF;
        void setMyF(F myF) {this.myF = myF;};
}
class H extends F {
}
```

```
public class Test {
        public static void main(String[] args) {
                A a = new A();
                B b = new B();
                C c = new C();
                D d = new D();
                E e = new E();
                F f = new F();
                G g = new G();
                H h = new H();

                a.setMyB(b);
                a.setMyB(e);
                a.setMyB(f);
                a.setMyB(d);
                a.setMyB(g);
                a.setMyB(h);
                g.setMyF(b);
                g.setMyF(f);
                g.setMyF(h);
        }
}
```

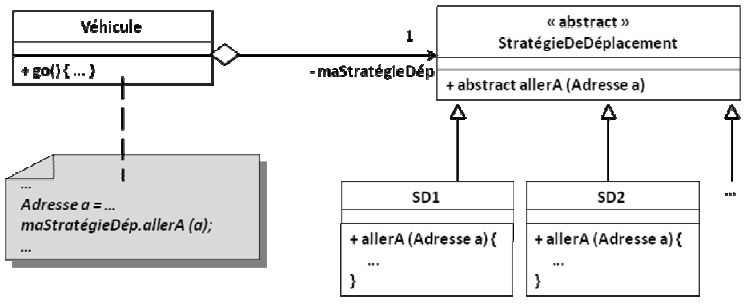
2) Donner un diagramme de classes (et les éléments de code utiles) qui supporte le programme suivant :

```
public class Test {
        public static void main(String[] argv) {
                Figure[] tfig = new Figure[3];
                tfig[0] = new Carré(Couleur.bleu, 2); // Création d'un carré bleu de 2 cm de coté
                tfig[1] = new Cercle(Couleur.vert, 3); // Création d'un cercle vert de 3 cm de rayon
                tfig[2] = new Carré(Couleur.noir, 5); // Création d'un carré noir de 5 cm de coté
                for (int i = 0; i < tfig.length; i++) {
                        System.out.println(tfig[i].toString() +" "+ tfig[i].getCouleur()+" " + tfig[i].aire());
                }
        }
}
```

Sur quel concept fondamental repose l'exécution de ce programme ?

EXERCICE 2

1) Quelle est la principale différence entre Patron de Méthode et Stratégie ?
2) Le diagramme de classes ci-dessous définit une application basée sur le design pattern Stratégie : la classe Véhicule utilise une stratégie de déplacement dont le supertype est StratégieDeDéplacement.



Supposons que les différentes implantations de la méthode allerA(Adresse a) dans les classes SD1, SD2… ne varient que pour une petite partie : le calcul de l'itinéraire.

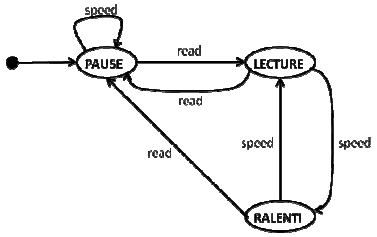
a. Quel design pattern convient pour éviter la duplication du code commun ?
b. Critiquez les différentes mises en œuvre au moyen de ce pattern qui sont proposées par Alice, Bob, Carole et David (voir en annexe).
c. Concevoir la solution en modifiant le diagramme de classes ci-dessus. Donner les éléments de code utiles.
d. Donner les quelques lignes de code qui réalisent la création et la configuration d'un véhicule (avec une stratégie au choix) puis l'invocation de la méthode go(). Pour cela, il faut expliciter le constructeur de Véhicule et, éventuellement, une méthode d'affectation de la stratégie de déplacement (setter).

Avertissement (démarche) : Pour répondre aux problèmes de conception qui suivent, on peut soit utiliser les design patterns déjà étudiés, soit concevoir la solution ex nihilo. En d'autres termes, on ne doit pas consulter le « catalogue » dans son intégralité, mais seulement les descriptions des design patterns déjà étudiés.

EXERCICE 3

On considère un objet de type Lecteur qui permet de lire une vidéo. On suppose que la vidéo peut être lue en boucle (elle n'a pas de début ni de fin). A tout instant, le Lecteur est :

- soit en PAUSE
- soit en cours de LECTURE en mode normal
- soit en cours de lecture en mode RALENTI



Deux actions, READ et SPEED, sont possibles sur le Lecteur. Pour cela, le Lecteur offre deux méthodes sans paramètre read() et speed() dont le résultat est void. L'effet de l'action dépend de la situation :

- l'action READ, quand le Lecteur est en PAUSE, provoque la LECTURE de la vidéo en mode normal
- l'action READ, quand la vidéo est en cours de LECTURE en mode normal ou en mode RALENTI, met le Lecteur en PAUSE
- l'action SPEED, quand le Lecteur est en LECTURE en mode normal, fait passer en mode RALENTI et inversement
- l'action SPEED est sans effet quand le Lecteur est en PAUSE.

1) Concevoir l'application et décrire la solution (diagramme de classes, etc.).
2) La solution met en œuvre un design pattern. Le décrire. De quel autre pattern est-il proche et en quoi diffère-t-il ?

EXERCICE 4

Le design pattern Proxy est un design pattern structurel de niveau objet. Il a pour objectif la conception d'un objet P (le proxy) qui se substitue à un objet S (le sujet) de manière transparente pour les clients, et qui contrôle l'accès à S.

Les questions 1 et 2 ont pour objectif de retrouver le design pattern Proxy puis de le décrire (précisément les parties « problème et contexte » et « solution »).

1) Donner une description du problème auquel répond le design pattern Proxy (intention, motivation, indications d'utilisation).
2) Décrire la solution : diagramme de classes, participants (et leurs rôles), collaborations.

Il existe différentes formes de proxies. On s'intéresse ici au « proxy virtuel » qui permet la création paresseuse du sujet (réel) : le sujet (réel) est créé par le proxy quand ce dernier est invoqué pour la première fois ; une fois le sujet (réel) créé, le proxy lui fait suivre tous les appels. Le comportement du proxy dépend donc du fait que le sujet existe ou pas.

3) Mettre en œuvre la solution.
4) Quel pattern pourrait être utilisé ici en association avec le pattern Proxy ? Discuter.
   Mettre en œuvre la solution au moyen de ce pattern que l'on composera avec le pattern proxy. Pour cela, on pourra modifier la solution structurelle du proxy. Donner le diagramme de classes, les parties de code utiles et tous les éléments de documentation nécessaires.