Analyse et Conception Orientée Objet – ING1

TD 9 : Conception Orientée Objet - CORRIGE

Il s'agit dans ce travail dirigé d'introduire :

- le concept d'interface pour privilégier la programmation générique à la programmation spécifique
- la réutilisation : héritage ou composition ?

Exercice 1. Interface : Un dessin et des figures

Il s'agit dans cet exercice de concevoir un logiciel de dessin. Un dessin est formé de figures. Une figure a un nom et une forme. Pour l'instant, on sait qu'on utilisera les formes suivantes :

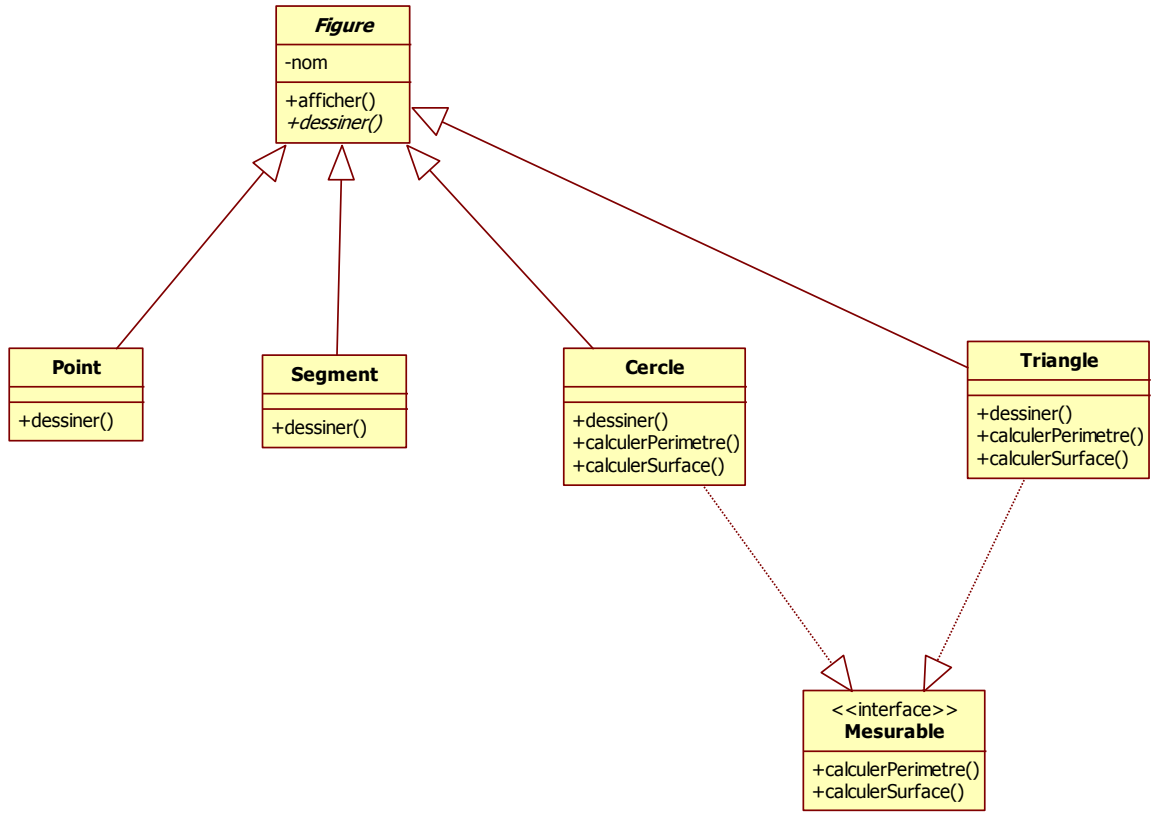
- des triangles;
- des cercles;
- des segments;
- des points.

Afficher une figure du dessin consiste à :

- dessiner la forme;
- afficher son nom;
- afficher son périmètre et sa surface si cela a un sens.

Identifier les classes concrètes, les classes abstraites et les interfaces. On justifiera sa réponse.

Corrigé :



Exercice 2. Réutilisation : Confusion entre héritage sémantique et héritage fonctionnel

On essaie dans cet exercice de montrer que l'héritage n'est pas toujours une bonne solution pour la réutilisation. Dans le package java.util la classe Stack a été implémentée en dérivant la classe Vector.

1) Expliquer en quoi le code suivant :

```
Stack pile = new Stack();
pile.push("Bas de la pile");
pile.push("Haut de la pile");
pile.insertElementAt("Perdu", 0);
while (!pile.empty()) {
      System.out.println(pile.pop());
}
```

viole le fonctionnement d'une pile.

Corrigé :

Les deux premiers éléments ont été placés dans la pile avec le principe LIFO mais pas le troisième. Quand on exécute le code on a l'affichage suivant :

Haut de la pile Bas de la pile Perdu

alors qu'on aurait du avoir

Perdu Haut de la pile Bas de la pile

2) En quoi a-t-on fait un héritage fonctionnel et non pas un héritage sémantique. Pourquoi ne peut-on pas dire qu'une pile **est un** vecteur ?

Corrigé :

On a utilisé la réutilisation de la classe Vector en tenant compte de certaines des ce fonctionnalités comme l'ajout d'un élément, la suppression d'un élément et la récupération d'un élément. Le principe LIFO a été pris en compte dans l'API Stack mais n'est pas respecté dans l'API Vector.

Une pile n'est pas un vecteur car

- on ne peut pas ajouter un élément à n'importe quel endroit : la méthode **add(int index, E element);**
- on ne peut pas supprimer des éléments dans un ordre quelconque : la méthode **remove(int index);**

3) Proposer une implémentation de la classe Pile en utilisant la classe Vector par composition.

```
import java.util.*;
class GoodStack {
//les signatures des méthodes sont comme celles de la classe Stack de Java
      private Vector v;
      public GoodStack() {
            v = new Vector();
      }
      public boolean empty() {
            return v.isEmpty();
      }
      public Object peek() {
            return v.lastElement();
```

```
	}
	public Object push(Object o) {
		v.add(o);
		return o;
	}
	public Object pop() {
		if( ! empty() ) {
			Object o = v.lastElement();
			v.remove(o);
			return o;
		}
		else
			return null;
	}
	public int search(Object o) {
		return v.indexOf(o);
	}
}
```

Exercice 3. Réutilisation : La bonne utilisation de l'héritage

On montre dans cet exercice que l'héritage est un bon choix pour la réutilisation quand la relation est de la forme : ObjetFille **est un (ou est une sorte de)** ObjetMere. On suppose que la classe Vehicule a été implémentée (définition : *un véhicule est un moyen qui permet à un ou plusieurs être humains de se déplacer d'un endroit A à un endroit B*).

1) Quels sont les liens sémantiques entre un véhicule et
- une voiture, une moto, un vélo, un cheval, un éléphant, un train, un avion, un bateau et un radeau
- un chauffeur

Corrigé :

Une voiture, une moto, un vélo, un cheval, un éléphant, un train, un avion, un bateau et un radeau **sont des sortes** de véhicules.

Un chauffeur **utilise** un véhicule.

2) On suppose que dans la classe Vehicule on a implémenté l'opération **void deplaceToi()**. On constate un peu plus tard que le déplacement dépend du vent et que la signature devient **void deplaceToi(int vent)**. Que faut-il recoder si :
   - si on a réutilisé par héritage la classe Vehicule pour les classes Voiture, Moto, Velo, Cheval, Elephant, Train, Avion, Bateau et Radeau.
   - si on a réutilisé par composition la classe Vehicule pour les classes Voiture, Moto, Velo, Cheval, Elephant, Train, Avion, Bateau et Radeau.

Corrigé :

Par héritage, il suffira de recoder l'opération deplaceToi une seule fois dans la classe Vehicule. Par composition, il faudra recoder l'opération deplaceToi dans chacune des classes Voiture, Moto, Velo, Cheval, Elephant, Train, Avion, Bateau et Radeau.

3) Qu'est ce qui nous assure dans ce cas que l'héritage est une bonne solution ?

Corrigé :
Le lien qui relie la classe Vehicule d'une part et les classes Voiture, Moto, Velo, Cheval, Elephant, Train, Avion, Bateau et Radeau d'autre part est un lien sémantique. Tout changement de comportement de la classe Vehicule doit être automatiquement reporté sur les classes Voiture, Moto, Velo, Cheval, Elephant, Train, Avion, Bateau et Radeau.

4) Reprenons le lien sémantique **un chauffeur utilise un véhicule**. Cela a pour conséquence qu'un véhicule peut ne pas avoir de chauffeur. On peut effectivement envisager qu'un véhicule se déplace automatiquement sans chauffeur. Expliquer pourquoi la composition est un meilleur choix.

Corrigé :
Par composition un véhicule contient une référence sur un chauffeur. Cette référence pourra donc être nulle. Si la classe Vehicule hérite de la classe Chauffeur alors un objet Vehicule contiendra un objet Chauffeur même dans le cas où il n'y a pas de chauffeur.