Université d'Aix-Marseille III - Licence de math-info 2ème année

I4 : Programmation objet - TD n°5 : classes abstraites et interfaces

1. Exercice d'examen de juin 04

On veut modéliser un ensemble de *composants électriques* : des *lampes*, des *rallonges* de fil électrique et des *prises multiples*. Les rallonges et les prises multiples sont considérées comme des *sources électriques* dans la mesure où on peut leur *rajouter* ou *retirer* en sortie (sur leur(s) prise(s) femelle(s)) un composant électrique (pour la rallonge) ou plusieurs (pour la prise multiple). Chacun des trois types de composants électriques peut *brancher* son unique entrée (sa prise mâle) sur une source électrique ou sur une prise murale (femelle). Son entrée peut aussi se *débrancher*. Chaque composant électrique est dans un des deux états suivants : soit sous tension, soit hors tension, suivant s'il est (directement ou indirectement) raccordé ou pas à une prise murale. Lorsqu'un composant électrique change d'état, il *propage* son état au(x) composant(s) electrique(s) qu'il a en sortie. Si c'est une lampe, elle ne propage rien et se contente d'afficher qu'elle s'allume ou qu'elle s'éteint. Un composant devient sous tension dès qu'il est branché à une prise murale ou à une source électrique sous tension ou parce que le composant auquel il est relié en entrée vient de passer sous tension. Inversement, un composant devient hors tension dès qu'il est débranché d'une prise murale ou d'une source électrique sous tension ou parce que le composant auquel il est relié en entrée vient de passer hors tension. Pour simplifier, on considérera que les lampes n'ont pas d'interrupteur et qu'elles sont allumées si elles sont sous tension.

Le but final de l'exercice est de définir des classes qui permettent par exemple l'exécution de la suite d'instructions suivantes :

```
Lampe l1 = new Lampe("Lampe1");
Lampe l2 = new Lampe("Lampe2");
Rallonge r = new Rallonge();
PriseMultiple p = new PriseMultiple(2);

l1.brancheSur(p);
l2.brancheSur(p);
p.brancheSur(r);
r.brancheSurPriseMurale(); // affiche "Lampe1 allumée" puis "Lampe2 allumée"
l1.debranche(); // affiche "Lampe1 éteinte"
l1.brancheSur(p); // affiche "Lampe1 allumée"
r.debranche(); // affiche "Lampe1 éteinte" puis "Lampe2 éteinte"
```

Après certaines instructions, on a mis en commentaire l'affichage qui était déclenché indirectement par l'instruction.

1.1 *Décrire et justifier une hiérarchie de classes permettant de modéliser au mieux les composants électriques tels qu'ils ont été décrits et de façon à ce qu'on puisse exécuter la suite d'instructions donnée ci-dessus. Plus précisément, il vous est demandé de donner l'ensemble des classes à définir, les liens (d'héritage ou de composition) entre les classes et la liste des méthodes (dont certaines sont éventuellement abstraites) de chacune de ces classes. Dans cette question, on ne demande pas*

d'écrire l'implémentation des méthodes ni les attributs des classes. La qualité de la conception de votre hiérarchie de classes sera évaluée, pas uniquement le fait que votre modélisation permet de mettre en oeuvre l'exemple donné ci-dessus.

Indices : les seules classes concrètes nécessaires sont `Lampe`, `Rallonge` et `PriseMultiple`. Ne pas déclarer de classe pour modéliser une prise murale (cf exemple). Une des possibilités de bonne modélisation de votre hiérarchie implique l'existence d'une classe abstraite et d'une interface.

1.2 *Donner les attributs des classes définies dans la question précédente et implémenter leurs méthodes.*

2. Exercice d'examen de septembre 04

On va créer des classes permettant de jouer à la bataille navale. Sur la *mer*, on dispose un certain nombre de *bateaux* dans un sens horizontal ou vertical en indiquant les coordonnées (entières) de leur coin supérieur gauche. Il y a quatre types de bateaux : le *porte-avions*, le *croiseur*, le *cuirassé* et le *torpilleur*, occupant respectivement 5, 4, 3 et 2 cases. Lorsqu'un joueur a joué un coup en indiquant des coordonnées, un bateau peut être *touché* si une des cases qu'il occupe a ces coordonnées et peut être *coulé* si toutes ses cases ont été touchées. Après chaque coup, le programme doit afficher "Touché" si un bateau est touché, "Coulé" si c'était la dernière case à toucher pour le couler, et "Gagné" si c'était le dernier bateau à couler. Voici un exemple de déroulement de programme :

```
Mer mer = new Mer(2); // 2 bateaux à couler
// place un torpilleur horizontalement en (1,1) :
mer.ajouteBateau(new Torpilleur(1, 1, true));
// place un cuirassé verticalement en (3,2) :
mer.ajouteBateau(new Cuirasse(3, 2, false));
mer.coup(10,10); // n'affiche rien
mer.coup(1,1); // affiche : Touché
mer.coup(2,1); // affiche : Touché Coulé
mer.coup(3,2); // affiche : Touché
mer.coup(3,3); // affiche : Touché
mer.coup(3,4); // affiche : Touché Coulé Gagné
```

2.1 *Décrire et justifier une hiérarchie de classes permettant de modéliser au mieux la mer et les bateaux tels qu'ils ont été décrits et de façon à ce qu'on puisse exécuter la suite d'instructions donnée ci-dessus. Plus précisément, il vous est demandé de donner l'ensemble des classes à définir, les liens (d'héritage ou de composition) entre les classes et la liste des méthodes privées ou publiques (dont certaines sont éventuellement abstraites) de chacune de ces classes.* Dans cette question, on ne demande pas d'écrire l'implémentation des méthodes ni les attributs des classes. La qualité de la conception de votre hiérarchie de classes sera évaluée, pas uniquement le fait que votre modélisation permet de mettre en oeuvre l'exemple donné ci-dessus.

2.2 *Donner les attributs des classes définies dans la question précédente et implémenter leurs méthodes.*