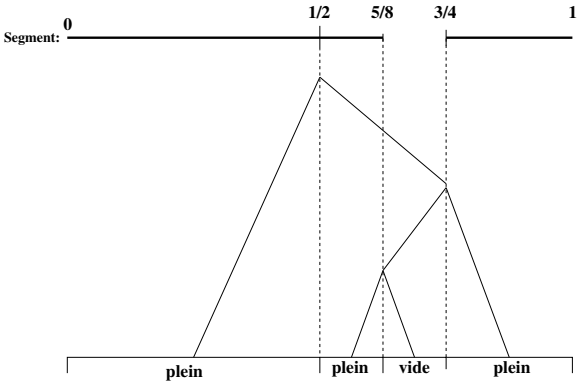
01234567890123456789012345

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

IN301 : Programmation objet - TD n°5 : classes abstraites et interfaces

1. (Exercice d'examen de janvier 08) On veut faire un programme qui modélise des segments potentiellement discontinus. Un tel type de segment est modélisé par la classe `Segment`. Voici comment on les définit (cf figure):

- soit c'est un unique segment continu (classe `SegmentContinu` en Java) qui est soit plein (classe `SegmentPlein`), soit vide (classe `SegmentVide`).
- soit c'est un segment discontinu (classe `SegmentDiscontinu`) et il est composé de deux segments (potentiellement discontinus) de longueurs égales à sa moitié.



On souhaite pouvoir effectuer deux types d'opérations sur les segments:

- déterminer leur taux de remplissage. La méthode `float remplissage()` retourne le taux de remplissage (un flottant de l'intervalle [0;1]) du segment:
  - si c'est un segment plein, c'est 1.
  - si c'est un segment vide, c'est 0.
  - si c'est un segment discontinu, c'est la moyenne des taux de remplissage de ses deux moitiés (la somme des taux divisée par 2).
- simplifier un segment. La méthode `Segment simplification()` retourne un segment simplifié selon le principe suivant:
  - si c'est un segment continu, il n'y a rien à simplifier: la simplification est le segment lui-même.
  - si c'est un segment discontinu, on calcule récursivement `s1` et `s2`, les simplifications de ses deux moitiés. Ensuite, si `s1` et `s2` sont des segments continus de même type (vide ou plein), on retourne `s1` (identique à `s2`), sinon on retourne un nouveau segment discontinu composé de `s1` et de `s2`.

Une fois toutes ces classes définies, on veut pouvoir par exemple écrire cette suite d'instructions qui crée un segment correspondant à celui de la figure:

```
SegmentVide vide = new SegmentVide();
SegmentPlein plein = new SegmentPlein();
SegmentDiscontinu s = new SegmentDiscontinu(plein,
        new SegmentDiscontinu(new SegmentDiscontinu(plein, vide), plein));
System.out.println(s.remplissage());
Segment s2 = s.simplification();
```

1.1 *Déterminez le type (classe concrète, abstraite ou interface) de chacune des classes :* `Segment`, `SegmentContinu`, `SegmentPlein`, `SegmentVide` *et* `SegmentDiscontinu`. *Pour chacune des classes, indiquez s'il faut (re)définir les méthodes* `float remplissage()` *et* `Segment simplification()` *et si elles sont abstraites ou concrètes dans cette classe. Justifiez vos réponses.* On ne demande pas d'écrire le contenu des classes.

1.2 *Ecrivez en java l'ensemble des 5 classes décrites ci-dessus.*

2. (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.

2.1 *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.

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