01234567890123456789012345

# Université d'Aix-Marseille III - Licence de Math-Info 2$^e$ année

## I4 : Programmation Objet - Examen
## Durée : 1h30 - ni document, ni calculatrice autorisés

**Important** : dans toutes les classes que vous écrirez, tout attribut devra être déclaré privé.

**1. (12 points)** On veut faire un programme qui modélise les copies d'examen. Chaque copie comporte un certain nombre de feuilles d'examen. Il y a deux types de feuilles : les feuilles d'examen et les feuilles de brouillon. Chaque feuille comporte deux pages : un recto et un verso. Chaque page contient un numéro de page et un texte.

**1.1** *Ecrivez la classe* `Page` *qui modélise une page et définit les méthodes suivantes:*
- un constructeur prenant un numéro de page (un entier) en paramètre.
- la méthode `void ecris(String t)` qui ajoute le texte `t` à la page (en plus du texte déjà présent).
- la méthode `String lis()` qui retourne le texte présent sur la page.

**1.2** *Ecrivez la classe* `Feuille` *qui modélise une feuille.* Elle contient évidemment une page en recto et une page en verso et des méthodes d'accès à ses attributs. Cette classe n'est pas destinée à avoir des instances.

**1.3** *Ecrivez la classe* `Brouillon` *qui modélise une feuille de brouillon.* Elle hérite de la classe `Feuille`. Dans son constructeur sans paramètre, on fait en sorte que la première page soit numérotée 1 et que la deuxième page soit numérotée 2.

**1.4** *Ecrivez la classe* `FeuilleExamen` *qui modélise une feuille d'examen.* Elle hérite de `Feuille` et contient les méthodes suivantes:
- un constructeur prenant en paramètre le numéro `n` de la feuille. Il fait en sorte que la première page soit numérotée `2 * n - 1` et que la deuxième page soit numérotée `2 * n`.
- un constructeur prenant en paramètre le numéro `n` de la feuille ainsi qu'un brouillon `b`. Il effectue la même chose que le constructeur précédent mais en plus il recopie les deux pages de `b` sur la feuille.
- la méthode `void recopie(int p, Brouillon b, int pb)` qui écrit sur la page numéro `p` de la feuille ce qu'on peut lire sur la page numéro `pb` du brouillon `b`. Rappels: un numéro impair de page est un recto de feuille ; un entier impair modulo 2 égale 1.

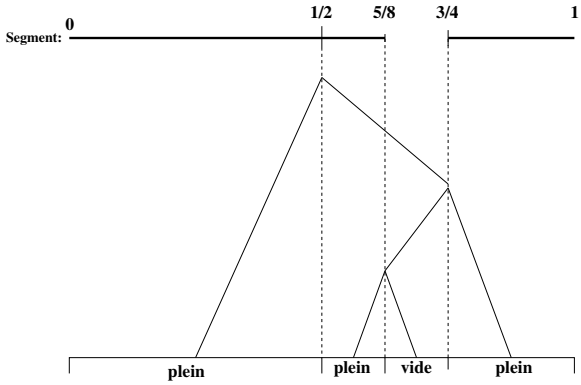
**1.5** *Ecrivez la classe* `Copie` *qui modélise une copie d'examen.* Au départ, elle contient deux feuilles mais on peut lui rajouter jusqu'à 4 feuilles. Elle mémorise aussi le nom de l'étudiant qui compose. Elle contient les méthodes suivantes:
- un constructeur prenant un nom d'étudiant (de type `String`) en paramètre. Il crée notamment les feuilles numéro 1 et 2.
- `void ajouteFeuille()` qui ajoute une feuille d'examen à la copie (s'il y en a moins que 6 au total).
- `void ajouteFeuille(Brouillon b)` qui ajoute une feuille d'examen à la copie en lui recopiant le contenu du brouillon `b`.
- `void ecris(int n, String s)` qui écrit `s` sur la page numéro `n`.
- `void ecris(int n, Brouillon b, int nb)` qui recopie la page numéro `nb` du brouillon `b` sur la page numéro `n` de la copie.
- `void relisTout()` qui affiche le contenu de chaque page de la copie dans l'ordre.

**1.6** On veut maintenant avoir les moyens de déterminer s'il y a eu triche. Pour ceci, on considère qu'il suffit que deux pages existantes aient un texte identique. *Modifiez la classe* `Page` *afin de rendre possible la*

*détection de triche, notamment en définissant la méthode de classe* `boolean testeTriche()` *qui compare deux à deux chaque paire de pages parmi toutes celles qui ont été créées.*

**2. (8 points)** 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();
```

**2.1** *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.

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