Université Paris Diderot — Année 2018-2019
L1 - Informatique — 8h40 - 10h30

Partiel IP2 Mercredi 6 mars 2019

Aucun document autorisé. N'utilisez **aucune des librairies de java** qui sont en rapport avec les listes.

Ce sujet est composé de 4 exercices traitants chacun une partie de ce que nous avons vu ensemble. Ils sont indépendants, et à mon avis de difficulté croissante. Le barème est indicatif, s'il évolue ce sera en donnant des bonus aux exercices traités complètement et correctement. Le temps estimé est indiqué pour vous donner un autre repère.

Exercice 1 (2 points) - (5 minutes)

Que signifie et à quoi sert chacun des mots clés suivants : **implements**, **final**, **this**, **static** ?

Répondez sans donner d'exemples, mais en trouvant une formulation qui montre que vous comprenez bien de quoi il s'agit.

Exercice 2 (5 points) - (15 minutes)

Un cambrioleur utilise des sacs pour transporter son butin constitué d'objets qui ont une valeur, un poids et un volume propre. Les sacs ont tous une contenance limitée, à la fois en volume (50 litres) et en poids. Si le premier sac peut supporter 20 kilos, chaque nouveau sac sera plus fragile de 10%. Le second ne supportera que 18 kilos, le suivant 16,2 kilos etc ...

Le voleur n'est pas limité dans la quantité des sacs qu'il utilise, et quand il juge que celui qu'il remplit est prêt, il le ferme et le lance par la fenêtre à un complice, puis il commence à en remplir un autre.

Tant qu'il y a de la place dans un sac il peut essayer d'y mettre les objets de son choix, mais il n'a pas le temps de faire attention s'il l'a surchagé ou non. Si c'est le cas il perdra l'ensemble de ce qu'il contient au moment où il voudra le transporter. Idem s'il a oublié de le fermer avant de le lancer.

N'écrivez pas de classe pour le cambrioleur. Ecrivez les classes Sac, Objet, et Test où vous illustrerez un exemple significatif de ses actions, en les commentant. Vous afficherez également le montant de ce qu'il aura réussi à voler, et de ce qu'il aura gâché (c'est à dire perdu accidentellement).

Exercice 3 (6 points) - (30 minutes) On s'intéresse à des listes simplement chaînées dont les cellules contiennent des char. D'une certaine façon ces listes s'apparentent donc à des mots, et c'est ainsi qu'on les désigne dans cet exercice. Les classes utilisées sont les suivantes :

```
public class CelluleChar{
  private char c;
  private CelluleChar next;
  public CelluleChar(char x, CelluleChar n){ c=x; next=n; }
}
public class Mot{
  private CelluleChar first;
}
```

Dans un mot, un caractère constitue un *bégaiement* s'il est identique au caractère suivant. On peut obtenir un mot *réduit* en supprimant tous les bégaiements. Deux mots ont *le même sens* si leurs mots réduits sont égaux. Nous répondrons à ces problèmes dans les questions suivantes.

1. **(1 point)** Ecrivez dans la classe CelluleChar une méthode **statique** qui prenne en argument deux CelluleChar et qui renvoie si oui ou non les séquences de caractères associées sont égales. (Vous pouvez librement choisir de l'écrire en récursif ou en itératif).

2. **(1.5 points)** Ecrivez une méthode d'objet de la classe Mot qui compte son nombre de bégaiements en utilisant **une approche itérative**.

3. **(1.5 points)** Ecrivez une méthode d'objets de la classe Mot qui compte son nombre de bégaiement en utilisant **une approche récursive**.

4. **(1.5 points)** Ecrivez une méthode d'objet (en itératif ou en récursif selon votre préférence) qui réduise le mot courant en supprimant tous les bégaiements.

5. **(0.5 points)** Ecrivez une méthode d'objet qui renvoie si deux mots ont le même sens.

Exercice 4 (7 points) - (30 minutes) On s'intéresse à des points du plan, repérés par leurs coordonnées x et y.
Pour deux points p et q on note que $p \leq_x q$ ssi la coordonnée x de p est inférieure ou égale à celle de q. De manière similaire on définit l'ordre $\leq_y$ selon la seconde coordonnée, y.

Pour modéliser un ensemble de points, et pouvoir les énumérer facilement dans l'ordre croissant selon $\leq_x$ ou $\leq_y$, on imagine une structure qui chaîne chaque point à la fois avec son successeur pour l'ordre $\leq_x$ **et** avec celui pour l'ordre $\leq_y$. Chaque point sera encapsulé dans une seule cellule, qui aura un successeur pour $\leq_x$ et, indépendamment, un autre successeur pour $\leq_y$. En cas d'égalité de coordonnée, l'important est que tous les points concernés apparaissent en respectant l'ordre, ce qu'on supposera réalisé.

La liste globale repère le plus petit point pour $\leq_x$ et le plus petit point pour $\leq_y$, pour pouvoir ainsi commencer les énumérations.

On vous donne le modèle à trois classes correspondant :

```
public class Point {
  private final int x,y;
  public Point (int a, int b) {x=a; y=b;}
  public int getX() {return x;}
  public int getY() {return y;}
}
public class CelluleOrdonneeXY{
  private final Point p;
  private CelluleOrdonneeXY nextX;
  private CelluleOrdonneeXY nextY;
  public CelluleOrdonneeXY(Point x, CelluleOrdonneeXY a, CelluleOrdonneeXY b){
    p=x; nextX=a; nextY=b;
  }
  public Point getPoint() {return p;}
  public CelluleOrdonneeXY getNextX() {return nextX;}
  public CelluleOrdonneeXY getNextY() {return nextY;}
  public void setNextX(CelluleOrdonneeXY c) {nextX=c;}
  public void setNextY(CelluleOrdonneeXY c) {nextY=c;}
}
public class ListeOrdonneeXY{
  private CelluleOrdonneeXY firstX;
  private CelluleOrdonneeXY firstY;
  public ListeOrdonneeXY(){ firstX=null; firstY=null;}
  public static ListeOrdonneeXY exemple() {...}
}
```

Les questions suivantes sont indépendantes.

1. **(1 point)** Ecrivez le code de la méthode exemple qui retourne une liste contenant les points $(0;2)$ et $(3;1)$.

2. **(1 point)** Dans ce contexte, expliquez le plus clairement possible pourquoi il est important d'avoir déclaré certains attributs finaux.

3. **(1.5 point)** Ecrivez de **manière itérative** des méthodes qui permettent d'afficher, dans l'ordre croissant des coordonnées x, l'ensemble des points contenus dans une de ces listes. Précisez bien dans quelles classes vous les écrivez. L'affichage se fera sur une seule ligne, avec un retour à la ligne à la fin.

4. **(1.5 point)** Ecrivez de **manière récursive** des méthodes qui permettent d'afficher, dans l'ordre croissant des coordonnées y, l'ensemble des points contenus dans une de ces listes. Précisez bien dans quelles classes vous les écrivez. L'affichage se fera sur une seule ligne, avec un retour à la ligne à la fin.

5. **(2 points)** Ecrivez soigneusement une méthode de ListeOrdonneeXY qui prenne en argument deux coordonnées, et qui ajoute une cellule contenant un nouveau point, tout en préservant les deux ordres.

   Conseil : après avoir traité les cas particuliers, écrivez une méthode auxiliaire dans CelluleOrdonneeXY pour placer votre cellule selon l'ordre $\leq_x$ seul, puis indépendamment, une autre méthode pour placer votre cellule selon l'ordre $\leq_y$, et combinez ces deux opérations proprement.