Université Paris Diderot Année 2018-2019

L1 - Informatique

Partiel IP2 Mercredi 17 avril 2019

Aucun document autorisé, merci de ranger vos téléphones.

L'ensemble des réponses peut tenir sur une copie double : faites en sorte de rendre un travail **propre et clair !** Une réponse brouillonne, confuse, difficilement déchiffrable sera très probablement considérée comme étant incorrecte.

Le sujet est composé de 3 exercices indépendants sur 4 pages. Le barème est indicatif.

Exercice 1 (5 points) On souhaite écrire une classe java pour modéliser de simples ballons gonflables.

Pour cet exercice vous devrez impérativement écrire ET justifier votre code en présentant séparément :

1. le code complet d'une classe **Ballon** avec tous les éléments syntaxiques qui permettent de répondre au mieux aux descriptions que vous trouverez ci-dessous.
2. un tableau synthétique, qui reprend les numéros des descriptions, auxquels vous associerez des explications en français qui éclairent les choix que vous avez fait dans votre code pour y répondre au mieux.

Si vous avez besoin d'écrire d'autres classes vous pouvez le faire, vous avez ici également le droit d'utiliser des librairies java quelconques. En particulier nous vous rappelons que pour obtenir un nombre aléatoire, la classe `Math` contient une méthode de signature `static double random()` qui retourne une valeur réelle positive, aléatoire dans l'intervalle $[0, 1[$

Descriptions	Numéro
Les ballons portent un message, ici une lettre, ils sont livrés individuellement, déjà gonflés, imprimés et bouchés par un noeud qu'on ne peut pas défaire.	D1
La pression maximale qu'ils peuvent supporter est uniformément de 10 bars,	D2
Lorsqu'ils sont neufs la pression intiale est supérieure à 70% de cette valeur maximale, (sans qu'on sache exactement combien).	D3
On définit un évènement extérieur d'usure, sa conséquence exacte est à priori inconnue, on sait juste que si l'usure intervient il y a une chance sur trois pour que le ballon éclate, et que sinon il se dégonflera un peu (entre 0 et 10%)	D4
Un second type d'évènement concerne tous les ballons qui existent (où qu'ils soient) c'est le passage du temps : dans ce cas l'usure précédente s'applique à tous.	D5

Exercice 2 (5.5 points) On vous donne les classes suivantes :

```
public class ListeLogement {
  private CelluleLogement first;
}
public class CelluleLogement {
  private final String rue;
  private int num_boite;
  private CelluleLogement suiv;
  public CelluleLogement(String r,int n, CelluleLogement s){
    rue=r;
    num_boite=n;
    suiv=s;
  }
  CelluleLogement getSuivant() {
    return suiv;
  }
  public boolean estSitue(String r, int n){
    return (num_boite==n && rue.equals(r));
  }
}
```

Elles sont utilisées pour modéliser le portefeuille immobilier d'une entreprise sous la forme de la liste des biens qu'elle possède. On en retient les adresses sous la forme d'un nom de rue et d'un numéro de boite aux lettre. On suppose que c'est suffisant pour distinguer les biens entre eux.

La liste est ordonnée, au sens où tous les biens de la même rue sont regroupés consécutivement, et qu'au sein de cet ensemble ils sont ordonnés par ordre croissant de leurs numéros de boite aux lettres. Remarquez qu'une entreprise peut posséder les numéros 5 et 10 de la rue Thomas Mann sans posséder les numéros intermédiaires.

1. **(1.5 points)** Ecrivez une méthode boolean decaleBoiteAuxLettre(String rue, int num, int dx) qui renomme les boites aux lettres des biens de la rue spécifiée dont les numéros suivent strictement le numéro *num*, pour cela on ajoutera la valeur *dx* à leurs numéros. (Seuls les biens de cette rue sont concernés). La valeur retournée exprime qu'un décalage a eu lieu.

2. **(1.5 points)** Ecrivez une méthode boolean supprime(String rue, int num) qui supprime le logement ayant cette adresse s'il existe. La valeur de retour exprime que la suppression a bien eu lieu.

3. **(0.5 points)** Ecrivez une méthode boolean estApres(String rue, int num) de la classe CelluleLogement qui exprime si oui ou nom le logement porté par la cellule se situe après (strictement) l'adresse donnée en argument.

4. **(2 points)** void ajoute(String rue, int num) qui ajoute un nouveau logement à l'adresse spécifiée. S'il y a déjà un logement occupant cette adresse vous décalerez le numéro de l'ancien.

Exercice 3 (10 points) Certaines questions sont dépendantes des précédentes, mais vous pouvez toujours supposer y avoir répondu si nécessaire.

On s'intéresse ici à des listes construites sur des cellules doublement chaînées, qui stockent des entiers strictement positifs. Elles gardent en mémoire les deux références vers la cellule de tête et la cellule de fin de la liste. En voici la trame :

```
public class ListeDouble {
 private CelluleDouble debut;
 private CelluleDouble fin;

 public ListeDouble(){ debut=null; fin=null; }
 public boolean isempty(){ return debut==null;}
 public void addTete(int x){
  if (isempty()) {
   debut=new CelluleDouble(x);
   fin=debut;
  } else debut=new CelluleDouble(x,debut);
 }
}
```

```
public class CelluleDouble {
 ...(1)... int DEFAULT=100;
 private final int x;
 private CelluleDouble next;
 private CelluleDouble prev;

 ...(2)... CelluleDouble(int a) {
  if (a>0) x=a; else x = DEFAULT;
  next=null;
  prev=null;
 }

 // La cellule construite avec le code suivant viendra se brancher a la cellule c :
 // - elle precedera c,
 // - et elle n'aura elle meme pas de predecesseur
 ...(3)... CelluleDouble(int a, CelluleDouble c) {
  /* on demandera d'en ecrire le code dans la question 2 */
 }
}
```

1. **(1.5 point)** Réécrivez les lignes comportant les marques (1), (2), (3), en complétant les pointillés pour qu'elles soient optimales en terme de conception/modélisation ? Donnez une justification rapide et claire.

2. **(1 point)** Ecrivez le code associé à la signature (3), conformément à la description. (petit bonus si vous l'écrivez en 3 instructions)

3. On vous donne la classe :

```
public class PaireListe {
  public ListeDouble a;
  public ListeDouble b;

  PaireListe(ListeDouble x, ListeDouble y) {a=x; b=y;}
}
```

Pour une cellule c quelconque qui fait partie d'une liste l, on peut naturellement concevoir un découpage en 3 listes, fragments de la liste d'origine : une première liste l_gauche constituée de toutes les cellules qui précèdent c strictement, une autre l_droite constituée de toutes les cellules qui lui succèdent strictement, et la troisième est réduite à la cellule c toute seule.

On suppose que tous les accesseurs/modifieurs utiles sont déjà écrits dans la classe CelluleDouble. Vous pouvez donc utiliser sans les réécrire : CelluleDouble getPrev(), CelluleDouble getNext(), void setPrev (CelluleDouble z), void setNext (CelluleDouble z), int getVal()

(2 points) Dans le contexte où la cellule c appartient bien à la liste courante, écrivez dans la classe ListeDouble une méthode public PaireListe cut(CelluleDouble c) , qui retourne la paire de listes l_gauche, l_droite comme définies précédemment. A la fin de l' exécution, la liste courante ne contiendra plus que le singleton c. Vous réutiliserez les mêmes cellules sans en créer de nouvelles.

4. Soient c une cellule, et l_gauche, l_droite les listes situées à sa gauche et à sa droite. On pose Sg et Sd les sommes respectives des valeurs contenues dans l_gauche et l_droite. Une cellule c est la **médiale** d'une liste, s'il s'agit de la première cellule pour laquelle $Sg \geq Sd$.

   (a) **(0.5 points)** Pour une liste qui contiendrait les entiers de 1 à 10 (dans cet ordre) quel serait l'entier porté par sa médiale ? Quelles sont alors les valeurs de Sg et de Sd ?

   (b) **(2.5 points)** Ecrivez la méthode CelluleDouble getMediale() de la classe ListeDouble avec éventuellement les méthodes auxiliaires utiles à sa résolution.

5. Le découpage précédent peut être utilisé pour produire un arbre à partir d'une liste : la médiale c sera prise pour racine, avec pour fils gauche et droit les arbres produits à partir des listes gauches et droites issues du découpage de la liste selon c et ainsi de suite.

   Voici les classes qui permettent de définir un arbre.

```
public class Arbre {
  NoeudArbre racine;
}
public class NoeudArbre {
  private int x;
  private NoeudArbre fg;
  private NoeudArbre fd;

  NoeudArbre(int val, NoeudArbre a, NoeudArbre b) { x=val; fg=a; fd=b; }
}
```

   (a) **(0.5 points)** Dessinez l'arbre que l'on obtiendrait par cette méthode à partir d'une liste qui contiendrait les entiers de 1 à 10 (dans cet ordre)

   (b) **(2 points)** Ecrivez un constructeur d'arbre qui prend en argument une listeDouble et qui implémente la méthode décrite.