11/05/2021 - 13h00-15h00

Initiation à la Programmation (IP2)

Rappel : vous disposez de brouillon pour réfléchir. Les réponses attendues tiennent en quelques lignes seulement : comprenez bien que cela ne vous prendra pas beaucoup plus de temps de n'écrire au propre que lorsque vous aurez les idées claires. Si en revanche vous rendez quelque chose de confus, de peu lisible, d'ambigu, le correcteur ne peut que le constater et ne pas vous donner de bénéfice pour des présentations approximatives.
Conseil méthodologique : appliquez vous à terminer un nombre éventuellement limité d'exercice mais faits soigneusement. Il vaut mieux en faire un peu moins complètement, plutôt que d'essayer de tout faire maladroitement. Le barème est indicatif. **Aucun document autorisé**.

Exercice 1 (3.5 points) On vous décrit un arbre un peu curieux : il possède un nid à son sommet ; il a aussi exactement un nid à l'étage juste inférieur (à distance 1 du sommet) ; il a encore exactement un nid quelque part sur l'étage juste inférieur (celui qui est constitué des noeuds à distance 2 du sommet) ; et ainsi de suite en respectant toujours cette règle jusqu'en bas : il y a toujours exactement un nid par étage de l'arbre et pas un de plus.

On veut écrire une méthode boolean arbreAnids() qui nous dise si un arbre quelconque est lui aussi dans le cas qu'on vient de décrire.

Voici les classes que nous utiliserons :

```
public class Noeud {
  private boolean nidIci; // indique si oui ou non le noeud porte un nid
  private int distanceSommet;
  private Noeud filsG;
  private Noeud filsD;
}
public class Arbre {
  private Noeud sommet;

  public boolean arbreAnid(){
    if (sommet==null) return false;
    int hauteur=sommet.nbEtage();  // pour la question 1
    sommet.setAllDistances(0);     // pour la question 2
    int [] nbNidsADistance=new int [hauteur]; // par defaut c'est initialise' a zero
    sommet.recensement(nbNidsADistance);  // pour la question 3
    ... a terminer en question 4
  }
}
```

1. **(1 point)** Ecrivez, uniquement dans la classe Noeud, une méthode int nbEtage() qui calcule le nombre d'étage qu'il y a en dessous de ce noeud (en comptant celui du noeud).

2. **(1 point)** Dans la classe Arbre, ligne 13 on fait l'appel sommet.setAllDistances(0). Ce qu'on veut c'est que cette méthode parcoure tous les noeuds en mettant à jour l'attribut distanceSommet pour s'assurer ensuite qu'il portera bien la valeur correcte de la distance de chaque noeud au sommet. Ecrivez cette méthode.

3. **(1 point)** Ecrivez ensuite la méthode présentée en ligne 15 : le recensement consiste à mettre à jour le tableau nbNidsADistance de sorte qu'après le recensement nbNidsADistance[i] indique le nombre de nids qui sont à la distance i du sommet.

4. **(0.5 point)** Ecrivez ce qu'il manque au niveau de la ligne 16 pour conclure la méthode arbreANids

Exercice 2 [4.5 points]

On vous donne les définitions syntaxiques suivantes :

```
public class A {
  private B tete;
}
public class B {
  private int x;
  private B next;
}
```

1. Pourvu que les éléments de A modélisent bien une structure de liste simplement chaînée (c'est à dire qui termine avec null), on peut imaginer le scénario suivant : un lièvre et une tortue commencent ensemble une course au niveau d'une tête de liste. La tortue avance pas à pas, de cellule en cellule, alors que le lièvre avance deux fois plus vite : d'abord la tortue, puis le lièvre etc... Ainsi, lorsque le lièvre franchit la ligne d'arrivée, la tortue dépasse seulement la moitié du chemin.

   (a) **[1 point]** Ecrivez une méthode de la classe A : void affichePastMilieu() qui implémente ce scénario de manière itérative. La valeur affichée correspondra à l'entier x de l'endroit où se trouve la tortue lorsque le lièvre arrive. Vous écrirez **une** méthode auxiliaire dans B et **aucun** accesseur. L'ensemble du code ne doit pas dépasser 15 lignes (propre et sans ratures). Vous pouvez lire la question suivante pour vous aider à étudier quelques cas particulier.

   (b) **[1 point]** Justifiez ce que produit **votre** solution lorsque l'élément sur lequel on invoque la méthode:

   - est une liste vide
   - est une liste qui ne contient qu'un entier
   - est une liste qui contient $0, 1, 2, 3, 4$ dans cet ordre.

2. Naturellement la méthode précédente ne fonctionne pas dans le cas où l'élément de A ne modélise pas une liste simplement chaînée bien formée (parce qu'une de ses cellules B boucle vers une cellule précédente). Mais dans ce cas, lors une course, on peut dire que le lièvre dépasserait la tortue indéfiniment.

   Cette remarque est en fait un moyen de détecter si un élément de A est une liste chaînée bien formée ou pas : le lièvre dépasse t'il la tortue à un autre moment qu'au départ ?

   Voici le schéma de la réponse à cette question :

```
1  public class A {
2    ....
3    public boolean isListe() {
4      if (tete==null) return ... ;
5      return B.isListe(tete, tete);
6    }
7  }
8  public class B{
9    ....
10   public static boolean isListe(B tortue, B lievre) {
11     // une solution recursive fait entre 3 et 10 lignes de code
12   }
13 }
```

   (a) **[0.5 point]** Que faut-il écrire en ligne 4 ?

   (b) **[2 points]** Ecrivez le code de la méthode déclarée en ligne 10, de manière **récursive**. La solution tient en très peu de lignes (soyez propre et sans ratures)

Exercice 3 (4.5 points) On utilise ici un système à 3 tableaux d'entiers de même longueur $etiq$, fg, fd pour modéliser un arbre dont les noeuds sont étiquetés par des entiers. L'idée est que chaque indice i dans ces tableaux permette de caractériser un noeud différent : la valeur de son étiquette est donnée par $etiq[i]$, l'indice de son noeud fils gauche est caractérisé par $fg[i]$, celui de son fils droit $fd[i]$. Et par convention, lorsqu'il n'y a pas de fils correspondant, la valeur de $fg[i]$ (ou de $fd[i]$) est -1.

1. **(1 point)** Dessinez l'arbre que modélise ce système :

```
int [] etiq= {23, 2, 3,  5,  7, 11, 13, 37, 41, 19};
int [] fg =  {-1, 5, 3, -1, -1,  9, -1,  8,  6, -1};
int [] fd =  {-1, 4, 0, -1, -1, -1,  2,  1, -1, -1};
```

   On vous donne la trame de code suivante qui défini les noeuds et les arbres. Vous y trouverez une méthode build qu'il faudra compléter : elle prend en argument les 3 tableaux que nous venons de décrire et elle doit retourner l'arbre correspondant à ce système si c'est possible, et null sinon.

```
public class Noeud {
  private int val;
  private Noeud fg;
  private Noeud fd;
  public Noeud (int x, Noeud g, Noeud d) {
    val=x;
    fg=g;
    fd=d;
  }
  public Noeud(int x) {this(x,null,null);}
  public void setG(Noeud g) {fg=g;}
  public void setD(Noeud d) {fd=d;}
}

public class Arbre {
  private Noeud racine;
  private Arbre (Noeud n) {racine = n;}

  public static Arbre build(int[] etiq, int[] fg, int[] fd) {
    // Question 2 - Detection d'erreurs evidentes ou d'arbre vide
    ...
    // Question 3 - Creation des noeuds
    int nbNoeud= ...;
    Noeud [] all=new Noeud [nbNoeud];
    ...
    // Question 4 - Recherche de la racine + detection d'erreur
    boolean [] estFils =new boolean [nbNoeud];
    for (int i=0;i<nbNoeud;i++) estFils[i]=false;
    ...
  }
}
```

2. **(0.5+0.5 point)** En ligne 21 on s'intéresse aux cas d'erreurs triviales et évidentes dans les arguments qui empêchent de décoder davantage le système, mais aussi au cas où on souhaite modéliser l'arbre vide. Quel code écrire pour tenir compte de l'un ou de l'autre de ces cas ?

3. **(1 point)** à partir de la ligne 23, on décode le système en procèdant ainsi :
   - dans un premier temps on construit dans all autant de noeuds que nécessaire, en ne leur donnant que leurs valeurs d'étiquette et en fixant leurs fils gauches et droits à null
   - dans un second temps nous reprenons ces noeuds un par un pour leur donner les bonnes liaisons vers leurs fils.

   Ecrivez ces quelques lignes.

4. **(1.5 point)** En ligne 26 on va analyser tous les noeuds créés pour s'intéresser à deux questions :
   - y a t'il un noeud qui selon le système a deux pères ? (C'est alors un cas d'erreur qui empêche la construction de l'arbre)
   - y a t'il bien un seul noeud qui n'a pas de père ? (C'est alors la racine de l'arbre)

   Pour cela on choisi de marquer dans un tableau estFils le fait qu'un noeud a été déclaré fils d'un autre. Faites ce travail de marquage, et ce qui est nécessaire pour conclure la méthode build

Exercice 4 (7 points) On s'intéresse à une classe qui modélise des tickets de loto, dont voici la trame que vous compléterez dans les questions suivantes :

```
public class TicketLoto {
  // on se permet d'utiliser ici cette librairie java qui possede : add, get, size ...
  private final static List<TicketLoto> all=new LinkedList();
  private static int NB_Tickets=0;
  private static int NB_Tirages=0;

  private final int numDuTicket;
  private final int numDuTirage;
  private final int [] numerosChoisis;

  private TicketLoto (int [] props) {
    // Question 1
    ...
    // Question 2
    this.numDuTicket = ...
    this.numDuTirage = ...
    // Question 3
    ....
    // aussitot que le 100eme ticket est defini le tirage a lieu
    if (NB_Tickets==100) cloture();
  }

  public static TicketLoto buyTicket(int [] props) {
    if (checkFormat(props) && checkAllDifferent(props)) return new TicketLoto(props);
    else return null;
  }

  private static void cloture(){
    // d'abord un tirage aleatoire est fait
    ...
    // puis on determine le nombre et le numero des tickets gagnants
    ...
    // puis on remet tout en ordre pour pouvoir lancer la campagne suivante
    ...
    ...
    all.clear();  // methode de la librairie qui vide la liste
  }
}
```

Un joueur peut acheter un ticket de loto en utilisant la méthode buyTicket qui procédera à des vérifications avant de construire le ticket. Le joueur devra choisir à l'avance 6 chiffres différents, et transmettre son choix via l'argument props. Il espère que le tirage qui aura lieu plus tard lui donnera raison : le ticket sera gagnant si les chiffres qu'il y a fait inscrire sortent tous. Notez bien que l'ordre n'a pas d'importance : si on choisi $0, 1, 2, 3, 4, 5$ ou $5, 4, 3, 2, 1, 0$ on a des tickets équivalents.

1. **(1 point)** En ligne 13, il nous semble risqué d'écrire numerosChoisis = props expliquez pourquoi et proposez une alternative

2. **(0.5 point)** En ligne 15 et 16 on doit s'assurer que sur chaque ticket vendu il y aura un couple (numDuTicket,numDuTirage) qui permet de l'identifier sans ambiguïté. On pourra donc parler du *"3ème ticket vendu pour le 2ème tirage"*. Si vous allez un peu plus loin (ligne 20) vous pouvez constater qu'à l'achat du 100ème ticket du tirage en cours, une clôture se fera immédiatement. L'idée est que les numéros de tickets seront toujours de fait entre 0 à 99. Complétez ces 2 lignes.

3. **(0.5 point)** Ligne 18 : on veut ici mettre à jours la liste all pour qu'elle contienne tous les tickets vendus pour le tirage courant. Que faut-il écrire ?

4. **(0.5 point)** la méthode checkFormat doit permettre de vérifier que le tableau props est composé de 6 chiffres. Ecrivez là.

5. **(1 point)** la méthode checkAllDifferent vérifie que le tableau props est composé de nombres tous différents. Ecrivez là.

6. La méthode cloture se décompose en 3 étapes qu'il vous faut écrire :

   (a) **(1 point)** on détermine aléatoirement la combinaison des 6 chiffres différents qui seront considérés gagnants. (Vous pourrez utiliser Math.random() qui vous donne un double dans $[0; 1[$)

   (b) **(1.5 point)** on détermine le nombre et on affiche les numéros des tickets gagnants. Rq : pour parcourir all vous pouvez utiliser la notation for avec les : ou faire appel aux méthodes get(int index) et size()

   (c) **(0.5+0.5 point)** on effectue ici quelques opérations pour tout remettre en ordre et permettre à présent la vente des tickets pour le tirage suivant, écrivez les. Expliquez aussi pourquoi on a choisi de faire appel à clear pour vider la liste all, et qu'on n'a pas écrit pas all=new LinkedList())