Université Paris Diderot
L1 - Informatique

Année 2017-2018
8h30 - 11h00

Examen IP2 Vendredi 4 mai 2018

Aucun document autorisé

Ce sujet est composé de 4 exercices traitants chacun une partie de ce que nous avons vu ensemble. Ils sont indépendants, et de difficulté croissante. La stratégie la plus efficace pour cette épreuve consiste à répondre très proprement aux questions que vous aborderez en premier, ce qui devrait vous assurer la moyenne. Vous pourrez ensuite gérer le temps qu'il vous reste sur ce qui vous semble difficile. Le barème est indicatif.

Exercice 1 (5 points) (Modélisation)

Voici une ancienne version d'un ticket de jeu à gratter. Il s'agit d'un jeu de hasard pour lequel vous achetez un ticket, et qui peut vous rapporter une somme d'argent plus importante. La valeur de ce gain est dissimulée derrière une case à gratter.
Pour cet exercice vous devrez :

1. Ecrire une classe **Banco** avec tous les éléments (attributs, constructeurs, méthodes ...) qui permettent de répondre aux descriptions que vous trouverez ci-dessous. Ecrivez en le code complet.

2. Reporter sur votre copie les deux dernières colonnes du tableau suivant, et expliquer en français comment votre code répond à chacune des 8 descriptions. Si besoin, dites quelles méthodes vos avez écrites pour permettre d'en réaliser la fonctionnalité.

Descriptions	Numéro	Explications
Le prix de tous les tickets est uniformément de 1 € et est connu de tous	D1	
Le numéro de série permet de distinguer les tickets entre eux	D2	
Il y a deux cases à gratter qui masquent des entiers positifs : - celle marquée "Gain", - celle marquée "Nul si découvert" (qui est un nombre aléatoire à 3 chiffres)	D3	
On ne peut pas voir au travers de ces cases tant qu'elles n'ont pas été révélées (leur lecture retourne alors -1)	D4	
Une fois révélées, on ne peut plus les masquer	D5	
On peut les révéler indépendamment l'une de l'autre	D6	
Chaque valeur lisible est définitive, et déterminée dès le départ	D7	
Les probabilités de gains sont (voir annexe si besoin): Pour 75 % le ticket est perdant Pour 24 % le ticket rapporte 2 € Pour 1 % le ticket rapporte 1000 €	D8	

Annexe

Nom de classe	Méthode	Description
Math	static double random()	Retourne une valeur réelle positive, aléatoire dans l'intervalle $[0, 1[$

Exercice 2 (6 points) (Manipulation de cellules)

Le problème est le suivant, on dispose de deux horloges modélisées chacune par une liste circulaire [1]

- l'une **am** fonctionne le matin et est composée de cellules ayant tous les entiers de 0 à 12, dans cet ordre, sa tête est positionnée sur 0
- l'autre **pm** fonctionne l'après midi et ses cellules ont, de la même façon, pour contenu les entiers de 12 à 24, sa tête est positionnée sur 12

Pour une mauvaise raison chacune est un peu redondante (on pourrait se passer de 12 dans **am** et de 24 dans **pm**), et en fait ce que l'on veut vraiment utiliser c'est une horloge pour toute la journée. Voilà pourquoi on a besoin des deux méthodes suivantes pour les listes circulaires :

- **void retireLastCell()**
- **ListeCirc fusionneAvec(ListeCirc x)**

La première méthode retire la dernière cellule de la liste courante.

La seconde met "bout à bout" les deux listes circulaires, **en reprenant les mêmes cellules**.
A l'issue de cet appel, x et *this* deviennent des listes vides.

Ce code permettrait d'obtenir une horloge pour la journée, c'est à dire avec des chiffres allant de 0 à 23, tête positionnée sur 0, à partir de nos deux horloges précédemment décrites **am** et **pm**.

```
public class Test {
 public static void main(String [] args){
  MaListeCirc am = ...; // on suppose defini am (ne pas l'ecrire)
  MaListeCirc pm = ...; // on suppose defini pm (ne pas l'ecrire)
  am.retireLastCell();
  pm.retireLastCell();
  MaListeCirc horlogeJournee = am.fusionneAvec(pm);
  // on obtient une horloge pour la journee
  // am et pm deviennent vides
 }
}
```

- Question 1 : (2 points)

  Ecrivez uniquement la méthode **retireLastCell** pour le cas où on remplacerait **MalisteCirc** par **ListeCircDouble** donné ici :

```
public class ListeCircDouble {
 private Cellule2 tete;
}
public class Cellule2 {
 private int content;
 private Cellule2 next; // cellule suivante
 private Cellule2 prev; // cellule precedente
 public Cellule2 getNext(){ return this.next; }
 public Cellule2 getPrev(){ return this.prev; }
 public void setNext(Cellule2 x){ this.next = x;}
 public void setPrev(Cellule2 x){ this.prev = x;}
}
```

[1] Rappel : pour les listes circulaires, le long de la suite des cellules, l'une d'entre elle a pour successeur l'élément de tête.

- Question 2 : (2 points)

  Recopiez et complétez le code de la méthode **retireLastCell** donné ci dessous, pour le cas où on remplacerait **MalisteCirc** par **ListeCircS** :

```
public class ListeCircS {
 private CelluleS tete;
 public void retireLastCell(){
  // conditions particulieres
  ...
  ...
  // cas general
  CelluleS aux=this.tete;
  while(... != this.tete) {
   aux=aux.getNext();
  }
  // suppression
  ...
 }
}
public class CelluleS {
 private int content;
 private CelluleS next;
 public CelluleS getNext() { return this.next;}
 public void setNext(CelluleS x){ this.next = x;}
}
```

- Question 3 : (2 points) Ecrivez très soigneusement **fusionneAvec** dans le cas où les listes circulaires sont modélisées par les classes :

```
public class ListeCircDouble {
 private Cellule2 tete;
}
public class Cellule2 {
 private int content;
 private Cellule2 next;
 private Cellule2 prev;
}
```

Exercice 3 (5 points) (Arbres) On considère dans cet exercice des arbres binaires simples dont les noeuds portent des étiquettes entières (positives ou négatives). Les questions sont indépendantes.

- **(0.5 points)** Ecrivez les déclaration des attributs des classes utilisées pour modéliser les arbres.
- **(2 points)** Ecrivez une méthode d'affichage des valeurs portées par un arbre, correspondant à un **parcours en largeur**
- **(2.5 points)** Le *chemin* d'un noeud n à la racine r est la suite des noeuds, fils l'un de l'autre, qui les relient (extrémités comprises).

  Le *poids* d'un chemin est la somme des valeurs étiquetant ses noeuds.

  Ecrivez une méthode **public int nbCheminsPoids(int x)** de la classe **Arbre** qui retourne le nombre de noeuds dont le chemin à la racine est de poids x.

(Tournez la page)

Exercice 4 (5 points) (Récursion linéaire)

1. (2 points) Dans cet question on s'intéresse à des listes simplement chaînées contenant des entiers. Donnez une explication en français, la plus synthétique possible, de ce que calcule la méthode g. (On souhaite savoir si vous comprenez bien à quoi elle peut servir)

```
public class Cellule {
 private int content;
 private Cellule next;

 public int g(){
  return h(0,0,this.content);
 }

private int h(int a, int b, int c) {
  if (this.content < c) { b = a ; c = this.content; }
  if (this.next == null) return b;
  return this.next.h(a+1, b, c);
 }
}
```

2. (3 points) A l'inverse, lorsque les cellules considérées contiennent des caractères, on se pose le problème de savoir si une liste possède plus de 'a' ou plus de 'b'.

   On vous demande ici d'écrire, sans jamais utiliser ni de **while** ni de **for**, une méthode d'une classe **ListeChar** nommée **public char AouB()** qui :

   - retourne 'a' si la liste contient plus de 'a' que de 'b'
   - retourne 'b' si la liste contient plus de 'b' que de 'a'
   - retourne '-' si la liste contient autant de 'a' que de 'b'

   Indication : il vous faudra introduire une ou des méthodes auxiliaires.