Université Paris Diderot Année 2018-2019

L1 - Informatique

Examen IP2 Mardi 18 juin 2019 - durée 2h30

Faites en sorte de rendre un travail **propre et clair !** Une réponse brouillonne, confuse, difficilement déchiffrable sera considérée comme incorrecte. N'hésitez pas à utiliser une règle, des stylos de couleur, n'écrivez que ce qui est clair dans votre esprit.

Merci de ranger vos téléphones.

Aucun document n'est autorisé, vous trouverez une petite annexe en début de sujet.

Il y a 4 exercices indépendants. Le barème indicatif vous donne une idée du temps à consacrer aux exercices. Le premier porte sur la modélisation, le second sur l'implémentation de méthodes de listes chaînées, et les deux derniers sont des petits exercices sur les arbres.

Annexe

Voici un mémo de la documentation Java auquel vous pourrez vous référer en cas de besoin.

```
// dans java.lang.Math
static double random(); // Returns a double value >=  0 and < 1
```

```
//  dans java.util.Random
Random();               // Creates a new random number generator.
int nextInt(int n);     // Returns a random value between 0 (inclusive) and n (exclusive)
```

```
// dans java.util.List, pour des elements de type E
boolean add(E e);       // Appends the specified element to the end of this list
void add(int pos, E e); // Inserts the element e at the  position pos
void clear();           // Removes all of the elements from this list
boolean contains(E e);  // Returns true if this list contains the specified element
E get(int index)        // Returns the element at the specified position in this list
boolean isEmpty();      // Returns true if this list contains no elements.
int lastIndexOf(E e);   // Returns the index of the last occurrence of e, or -1 if none
E remove(int index);    // Removes the element at the specified position
boolean remove(E e);    // Removes the first occurrence of e, if it is present
int size();             // Returns the number of elements in this list.
```

Exercice 1 (7.5 points) On souhaite structurer les données d'un problème en créant trois classes pour modéliser le fait que des joueurs participent à des sports au sein d'équipes nationales.

Vous pouvez dans cet exercice utiliser les bibliothèques java.

Lisez bien les descriptions qui vous sont données, et travaillez d'abord au brouillon pour ensuite présenter votre travail au propre. La réponse demandée consiste en deux parties :

- d'une part le code complet des classes **Joueur**, **Equipe** et **Sport** avec les éléments syntaxiques les plus adaptés aux descriptions;
- et d'autre part un tableau synthétique qui reprendra les numéros des descriptions, auxquels vous associerez **en français** vos explications afin de lever toute ambiguïté sur vos interprétations. Justifiez en particulier les choix des mots clés java, l'introduction d'attributs, ainsi que les types choisis.

Voici les descriptions à respecter :

1. Un sport est reconnaissable à son nom (par exemple le volley, le basket, le football), et pour chaque sport on détermine la taille des équipes qui se trouvent sur un terrain.
2. Les joueurs sont identifiés par leurs nom, prénom et âge. Vous n'écrirez qu'un seul constructeur de joueur qui ne prendra en entrée que son nom et son prénom.
3. Seul l'âge des joueurs peut évoluer, et pour qu'il change le programmeur doit envoyer au joueur concerné un signal indiquant que sa date anniversaire vient de passer.
4. Une équipe est identifiée par un numéro unique, elle est associée à un seul sport, ainsi qu'à une nation.
5. Dans une équipe on distingue clairement les joueurs titulaires des joueurs remplaçants. Les titulaires sont ceux qui seront sur le terrain au début du match, leur nombre est strictement limité par la nature du sport, alors que le nombre de remplaçants est quelconque.
6. Il doit exister une façon de vérifier qu'une équipe a pour participant (au sens large) un joueur donné.
7. Lorsqu'un joueur voudra s'inscrire, la règle est que s'il reste encore des places sur le terrain alors il sera titulaire, sinon il sera remplaçant.
8. Il ne peut y avoir de redondance du même joueur dans une équipe donnée.
9. Par construction les joueurs savent dans quelles équipes ils sont inscrits, et les équipes savent quels joueurs y participent. Il faut donner à ces objets les moyens de stocker ces informations correctement.
10. Si un joueur se désinscrit d'une équipe et abandonne une place de titulaire, alors le plus jeune des remplaçants prendra sa place.

Exercice 2 (8 points)

«... L'univers ne connaît pas d'autre loi, il nous propose deux places, prédateur ou proie, deux positions aussi instables qu'interchangeables ... »[1]

On vous donne les classes suivantes :

```
public class Liste {
  private CelluleAnimal first;
  Liste() { first = null; }
}
public class CelluleAnimal {
  private Animal x;
  private CelluleAnimal suiv;
  public CelluleAnimal(Animal x, CelluleAnimal s){
    this.x=x;
    suiv=s;
  }
  public Animal getAnimal() { return x; }
  CelluleAnimal getSuivant() { return suiv; }
}
public class Animal {
  private final String nom;
  private int poids;
  public String getName(){ return nom; }
  public Animal (String n, int p){ nom=n; poids=p; }
}
```

Ecrivez les méthodes suivantes ainsi que les méthodes auxiliaires nécessaires que vous jugerez utiles :

1. **(2 points)** Dans la classe Liste : randomExemple(int x, String nomDeBase) qui retourne une liste constituée de x nouveaux animaux, de poids pris aléatoirement entre 1 et 10, et dont les noms sont construits à partir du nom de base suivit de leur numéro d'ordre de création. (Le premier créé doit rester en tête de liste, etc)

2. **(1 point)** Une méthode de la classe Animal : croque(Animal proie) qui permet à un animal considéré comme un prédateur de s'attaquer à une proie. Si le prédateur est au moins aussi gros que sa proie, alors il l'ingère et grossit du poids de son repas. Sinon la proie s'en sort indemne, et la méthode répond faux pour signifier que l'action a échoué.

3. Une liste d'animaux peut s'attaquer à une autre liste d'animaux. Dans ce cas on considère que, dans l'ordre de la liste des prédateurs, chacun ira croquer la première proie qu'il pourra effectivement manger. Si un prédateur ne trouve aucune proie, son poids sera divisé par deux (mais devra rester tout de même supérieur à 1).

   Pour illustrer ce comportement, on décompose le problème en :

   (a) **(2 points)** une méthode boolean croqueBy(Animal x) qui considère la liste courante d'animaux comme des proies dans laquelle x ira se servir. L'animal consommé devra disparaître, et le booléen indique que x a réussi à se nourrir ou non.

   (b) **(1 point)** une méthode public void attaque(Liste proies) de la classe Liste où la liste courante est vue comme la liste des prédateurs s'attaquant à des proies.

4. **(2 points)** Dans une classe Test, écrivez un main qui crée deux exemples de populations distinctes vivant l'une le jour, et l'autre la nuit. Au fur et à mesure que le soleil suivra sa course dans le ciel, la population active se nourrira de celle qui est inactive jusqu'à ce qu'il n'y ait plus qu'un seul groupe survivant. Complétez votre programme pour qu'il affiche le nom d'un de ces survivants.

[1] Éric-Emmanuel Schmitt

Exercice 3 (2 points)

Voici les classes qui permettent de définir des arbres dont les nœuds sont étiquetés par des caractères. Remarquez qu'ils ne sont pas binaires puisque les fils sont rangés dans un tableau.

```
public class ArbreN {
  NoeudArbreN racine;
}
public class NoeudArbreN {
  private char x;
  private NoeudArbreN [] fils;

  NoeudArbreN(char val, int nbFils) {
    x=val;
    fils = new NoeudArbreN[nbFils];
  }
  void setFils(int num, NoeudArbreN f) {
    fils[num]=f;
  }
  NoeudArbreN getFils(int num){
    return fils[num];
  }
}
```

Écrivez une méthode `int degreArbre()` de la classe `ArbreN` qui retourne le plus grand nombre de fils que peut avoir un nœud présent dans l'arbre.

Exercice 4 (3 points)

Voici les classes qui permettent de définir un arbre binaire dont les nœuds sont étiquetés par des chaînes de caractères.

```
public class Arbre2 {
  NoeudArbre2 racine;
}
public class NoeudArbre2 {
  private String x;
  private NoeudArbre2 filsG;
  private NoeudArbre2 filsD;
}
```

L'étiquette portée par un nœud x est considérée comme étant *insignifiante* lorsqu'il existe un nœud $y \neq x$ sur le chemin entre la racine et x qui porte déjà cette étiquette. Dans le cas contraire elle est *signifiante*. On veut compter le nombre d'étiquettes qui sont signifiantes dans un arbre.

1. Écrivez une méthode `int nbEtiquettesSignifiantes(List<String> etiquettesNoeudsSuperieurs)` dans la classe `NoeudArbre` qui calcule le nombre d'étiquettes signifiantes dans le sous-arbre issu du nœud courant, sachant que la liste des étiquettes portées par les nœuds situées entre la racine et le nœud courant est donnée en argument.

2. Écrivez une méthode `int nbValeursSignifiantes()` dans la classe `Arbre`.