I.U.T. Montpellier-Sète – Département informatique
Examen de *Structures de données et algorithmes fondamentaux* (programmation 1 bis - M1103)
lundi 4 janvier 2020
Barème excédentaire.
Pas de document ni de machine autorisés.

Gestion d'une forêt

Les fonctions doivent être écrites en langage Java. Elles peuvent faire appel aux fonctions écrites dans les questions précédentes. Le correcteur considèrera qu'une fonction appelée fonctionnera correctement, telle que spécifiée dans le sujet, même si vous n'avez pas écrit ou mal écrit le corps de cette fonction. Les initialisations doivent être faites explicitement, sans présumer que l'interprète Java initialise les variables aux valeurs 0 ou false. Il est inutile de recopier les commentaires (résultat, action, etc.) des fonctions à écrire. On rappelle qu'un pré-requis est une condition déjà vérifiée au début de l'algorithme ; inutile donc de tester les pré-requis dans les fonctions.

On considère l'évolution d'une forêt cultivée au cours du temps. Le temps est compté en années. Les événements sont les plantations et les coupes d'arbres dans la forêt.

1 Une classe `Arbre`

On définit d'abord une classe `Arbre` dont le seul attribut est l'âge (en années) d'un arbre :

```
public class Arbre {

   private int age;  // en années depuis la plantation de l'arbre, age >= 0
   ...
```

▷ **Q1** (1 pt.s.) Ecrire le constructeur : `public Arbre()`

```
// Action : construit un arbre au moment de sa plantation
```

▷ **Q2** (1 pt.s.) Ecrire la méthode : `public String toString()`

```
// Résultat : une chaîne de caractères de la forme « (âge de this) an »
//            si l'âge de this est 0 ou 1, et « (âge de this) ans » sinon.
```

▷ **Q3** (1 pt.s.) Ecrire la méthode : `public void passeAnnee()`

```
// Action : augmente l'âge de this d'une année.
```

▷ **Q4** (1 pt.s.) Ecrire la méthode : `public boolean ageDepasse(int limite)`

```
// Résultat : vrai ssi l'âge de this est supérieur ou égal à limite.
```

2 Une classe `Foret`

Vous allez maintenant écrire une classe `Foret`, définie par un « ensemble » d'arbres qui poussent dans la forêt :

```
public class Foret {
   private int nbArbres; // nbArbres >= 0
    // Nb d'arbres existants, c'est-à-dire déjà plantés et pas encore coupés, de la forêt.

   private Arbre[] tabArbre;
    // Tableau contenant les arbres existants de la forêt dans les cases d'indice 0 à
    // nbArbres - 1, rangés en ordre chronologique de plantation, et donc en ordre
    // d'âge décroissant.
   ...
```

▷ **Q5** (1,5 pt.s.) Ecrire le constructeur : `public Foret (int nbMax)`

```
// Action : construit une forêt sans arbre pouvant contenir au maximum nbMax arbres.
```

▷ **Q6** (1,5 pt.s.) Ecrire la méthode : `public String toString ()`

```
/** Résultat : chaîne de caractères qui donne à l'affichage l'état courant de la forêt
    sous la forme :
       arbre 0 : ... an (ou ans)
       arbre 1 : ... an (ou ans)
       arbre 2 : ... an (ou ans)
       etc.
    (on rappelle que le caractère de retour à la ligne est \n)
*/
```

▷ **Q7** (1 pt.s.) Ecrire la méthode : `public void passeUneAnnee ()`

```
 // Action : simule le passage d'une année de cette forêt.
```

▷ **Q8** (1 pt.s.) Ecrire la méthode : `public void planteUnArbre ()`

```
 // Pré-requis : this.nbArbres < this.tabArbre.length
 // Action : simule la plantation d'un arbre dans this.
 // Remarque : on rappelle que les arbres de la forêt apparaissent dans
 //      le tableau tabArbre dans l'ordre de leur plantation ;
 //      un nouvel arbre planté devra donc apparaître à droite de
 //      tous les arbres déjà présents.
```

▷ **Q9** (1 pt.s.) Ecrire la méthode : `public void plante (int nb)`

```
 // Pré-requis : 0 <= nb <= this.tabArbre.length - this.nbArbres
 // Action : simule la plantation de nb arbres dans this.
```

▷ **Q10** (2 pt.s.) Ecrire la méthode : `public void coupeVieuxArbres (int nb)`

```
 // Pré-requis : 0 <= nb <= this.nbArbres.
 // Action : simule la coupe des nb plus vieux arbres de this.
 // Remarque : le tableau tabArbre contient nb arbres de moins après la coupe.
 //            On rappelle que les arbres existants de la forêt doivent
 //            toujours figurer dans les cases d'indice 0 à nbArbres - 1
 //            de tabArbre.
```

▷ **Q11** (2 pt.s.) Ecrire la méthode : `private int nbArbresLimite (int limite)`

```
// Pré-requis : aucun.
// Résultat : retourne le nombre d'arbres de this dont l'âge est supérieur
//            ou égal à limite.
```

▷ **Q12** (2 pt.s.) Ecrire la méthode : `public int coupeArbresLimite (int limite)`

```
// Pré-requis : aucun.
// Action/résultat : simule la coupe des arbres de this dont l'âge est supérieur ou
//                   égal à limite, et renvoie le nombre d'arbres coupés.
```

3 Programme principal

▷ **Q13** (4 pt.s.) Ecrire la méthode : `public static void main (String[] args)` d'une classe `MainForet`

```
/** Action : simule la création d'une forêt sans arbre, puis, dans cette forêt,
 *    la plantation de 5 arbres par an de l'année 2020 à l'année 2080 (incluses),
 *    puis la coupe en 2081 de tous les arbres ayant au moins 50 ans,
 *    et la replantation en 2082 du même nombre d'arbres que celui des arbres coupés
 *    en 2081, et finalement affiche l'état de la forêt juste après cette replantation.
 *  Rappel : l'affichage en java se fera en utilisant "System.out.println(...)".
 */
```

4 Liste d'arbres

Nous voulons maintenant écrire une variante de la classe `Foret`, appelée `ForetListe`, qui utilise une liste chaînée pour obtenir la liste des arbres qui poussent dans cette forêt. L'attribut (unique) de la classe `ForetListe` est défini comme suit :

`private ListeArbre lisArbre`

où les arbres sont rangés en ordre d'âge décroissant dans la liste.

Il faut donc définir une nouvelle classe `ListeArbre` qui est une variante de la classe `Liste` vue en cours et dont les maillons de type `MaillonArbre` contiennent des valeurs de type `Arbre` (et non pas entier).

La classe `ListeArbre` possède un unique attribut privé `tete` de type `MaillonArbre` qui référence la tête de la liste (si la liste est non vide). La classe `MaillonArbre` est donnée en fin de sujet.

▷ **Q14** (1,5 pt.s.) Ecrire dans la classe `ListeArbre` la méthode : `public void supprimeTete()` :

```
// Pré-requis : this n'est pas vide.
// Action : supprime le premier élément de this.
```

▷ **Q15** (1,5 pt.s.) Ecrire dans la classe `ForetListe` la méthode :
`public void coupeVieuxArbres (int nb)`

```
// Pré-requis : 0 <= nb <= longueur de this.lisArbres.
// Action : simule la coupe des nb plus vieux arbres de this.
```

La classe `MaillonArbre`

```
public class MaillonArbre {

    private Arbre valeur;
    private MaillonArbre suivant;

    /** Constructeur vide */
    public MaillonArbre () {
       suivant = null;
    }
    /** Constructeur avec une valeur */
    public MaillonArbre (Arbre a) {
        valeur = a;
        suivant = null;
    }
    public Arbre getVal() {
        return this.valeur;
    }
    public void setVal(Arbre a) {
        this.valeur = a;
    }
    public MaillonArbre getSuiv () {
        return this.suivant;
    }
    public void setSuiv (MaillonArbre m) {
        this.suivant = m;
    }
    public String toString () {
        return this.valeur.toString();
    }
} // end class MaillonArbre
```