IMPORTANT. Durée **3 heures**. Il sera tenu compte de la **clarté** et de la **qualité** de votre copie. Aucun document autorisé.

1 Listes chaînées

Remarque : *Dans la suite, que ce soit pour les listes chaînées ou pour les arbres, on classe les méthodes par difficulté, il s'agit de difficulté liée à la compréhension, pas une mesure de la longueur du code.*

On considère les classes suivantes :

```
public class Noeud {

    private int val;
    private Noeud suivant;

    public Noeud(int x, Noeud n) {
        this.val = x;
        this.suivant = n;
    }
}
```

```
public class Liste {

    private Noeud tete;

    public Liste() {
        this.tete = null;
    }

    public void ajouter(int x) {
        this.tete =
              new Noeud(x, this.tete);
    }
}
```

Dans la suite, pour expliquer les méthodes, on représentera les listes par des suites d'entiers. Par exemple, $(1;4;5)$ repésentera une liste de 3 éléments dont le premier Noeud contient 1, le deuxième 4.... La liste vide sera représentée par $()$.

1.1 Questions

Les méthodes suivantes doivent êtres écrites dans `Liste`, certaines de ces méthodes nécessiteront de coder des méthodes auxilliaires dans `Noeud` ou dans `Liste`. Vous devez donc **impérativement** préciser dans quelle classe est codée chaque méthode.

1. (facile) Écrivez une méthode `int nbreZeros()` qui calcule le nombre de valeurs à O dans la liste sur laquelle est appelée la méthode. Par exemple, si `l` représente $(0;2;0;4;1)$, `l.nbreZeros()` doit retourner 2. Si la liste est vide, on retournera 0.

   Donnez deux versions de cette méthode : l'une devra faire appel à une méthode itérative, l'autre à une méthode récursive.

2. (facile) Écrivez une méthode `void enlevePremier()` qui enlève le premier élèment de la liste, si la liste est vide, il ne se passe rien. Par exemple, si `l` est $(1;2;5;4;1)$ après l'appel `l.enlevePremier();` `l` vaudra $(2;5;4;1)$.

3. (plus difficile) Écrivez une méthode **récursive** `void sommePartielle()` qui dans chaque noeud de la liste va mettre la valeur de la somme de tous les éléments qui sont après lui,

lui-même compris. Par exemple, si la liste avant application de la méthode est $(4; 1; 2; 5)$ la liste après application est $(12; 8; 7; 5)$, où par exemple 8 est bien la somme de 1, 2 et 5.

Indication : on utilisera le fait que la nouvelle valeur d'un élément est son ancienne valeur + la somme de l'élément suivant.

2 Arbres

On considère les classes suivantes :

```
public class Arbre {

    private NoeudArbre racine;

    public Arbre () {
        this.racine = null;
    }
}
```

```
public class NoeudArbre {

    private int etiquette;
    private NoeudArbre gauche;
    private NoeudArbre droit;

    public NoeudArbre(int etiquette,
                      NoeudArbre gauche,
                      NoeudArbre droit) {
        this.etiquette = etiquette;
        this.gauche = gauche;
        this.droit = droit;
    }

    public NoeudArbre(int etiquette) {
        this.etiquette = etiquette;
        this.gauche = null;
        this.droit = null;
    }
}
```

2.1 Questions

Les méthodes suivantes doivent êtres écrites dans `Arbre`, certaines de ces méthodes nécessiteront de coder des méthodes auxilliaires dans `NoeudArbre` ou dans `Arbre`. Vous devez donc **impérativement** préciser dans quelle classe est codée chaque méthode.

1. (facile) Écrivez une méthode `int etiquetteDroite()` qui retourne la valeur du nœud la plus à droite, c'est-à-dire celui qu'on obtient en allant à droite depuis la racine, le plus de fois possible. Si l'arbre est vide, on retournera -1. Par exemple, la méthode appliquée à l'arbre de la figure 1 donnera 1.
2. (plus difficile) Écrivez une méthode `void ajouteFeuille(int n)` qui à chaque feuille ajoute une feuille comme fils gauche et une feuille comme fils droit, ces nouvelles feuilles seront étiquetées par `n`. Si l'arbre est vide, on ajoutera une unique feuille étiquetée par `n`. Par exemple, la méthode appliquée à l'arbre de la figure 1 donnera l'arbre de la figure 2.

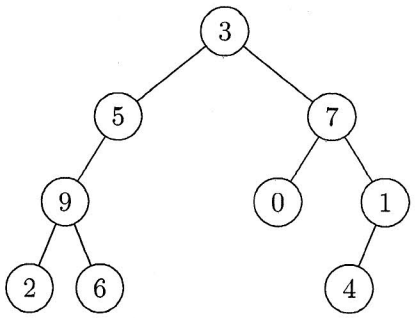

FIGURE 1 –

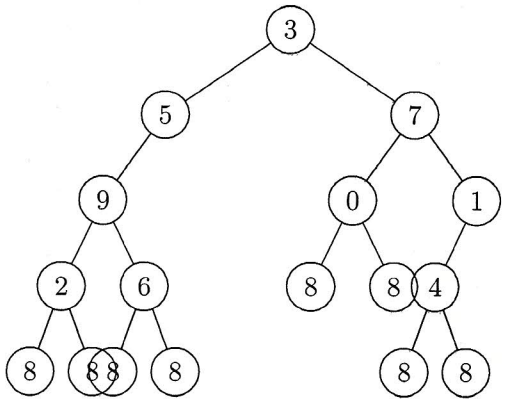

FIGURE 2 –

3 Exercice sur les objets

On s'intéresse à la modélisation d'un parc d'imprimantes. Chaque imprimante possède une marque, et un numéro de série. Elles sont chargées avec une certaine quantité de feuilles de papier, dont le nombre est inférieur à une limite propre à l'imprimante. Ces imprimantes contiennent un emplacement destiné à recevoir une cartouche d'encre. Ces cartouches sont elles mêmes des objets complexes : elles ont une contenance, exprimée en millilitres (un entier), une valeur indiquant le taux de remplissage (un pourcentage, c.à.d un entier entre 0 et 100), un indicateur qui précise deux états possibles de la cartouche : en bon état, ou inutilisable, et une description de la couleur de l'encre.

1. Ecrivez les attributs de deux classes qui modélisent respectivement les cartouches et les imprimantes.
2. Lorsqu'on fabrique une imprimante, celle ci est livrée sans feuille et sans cartouche. Lorsqu'on fabrique une cartouche, celle ci est pleine. Définissez un constructeur pour les cartouches, et un pour les imprimantes (Proposez une solution pour faire en sorte que les numéros de séries soient attribués de manière automatique tout en étant tous différents)
3. Lorsqu'on veut mettre un bloc d'un certain nombre de feuilles dans une imprimante, il se peut qu'on en remplisse complètement le bac avec une partie seulement du bloc, et qu'il en reste donc un certain nombre qu'on ne peut pas y loger. Ecrivez une méthode de remplissage de feuilles, qui prenne en argument un nombre de feuilles et qui retourne la quantité restante (celles qu'on n'a pas réussit à loger). Où écrivez vous cette méthode ? Quel est son type ?
4. De la même façon que pour les feuilles, il est toujours possible d'essayer de recharger une cartouche c'est à dire d'y introduire un certain volume d'encre. Pour les feuilles, on avait choisi de garder le surplus, malheureusement pour l'encre les choses ne sont pas aussi faciles, et si l'on ajoute trop d'encre, ou une encre d'une mauvaise couleur, alors on considère que la cartouche devient inutilisable. Ecrivez une méthode pour simuler ce comportement, justifiez soigneusement son type, ses arguments, et la classe où vous l'écrivez.
5. Ecrivez une classe Test contenant un `main`. Dans ce `main` fabriquez une imprimante et une cartouche, et faites appel aux méthodes que vous avez écrites.