Programmation Orientée Objet (Java)

TP n°1

Partie 1 : Liste chaînées

Exercice 1 : La base

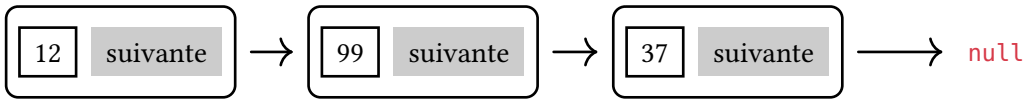

Fig. 1. – Structure d'une liste chaînée.

Le but de cet exercice est d'implémenter une version simplifiée des listes chaînées non-vides.

Comme décrit dans la Fig. 1 il s'agit d'une structure `Cellule` contenant un entier nommé `valeur` et une autre `Cellule` nommée `suivante`.

Question 1 : Construction de la classe

Créez la classe décrivant une telle structure.

Question 2 : Constructeurs

Ajoutez les constructeurs suivants :

- `public Cellule(int valeur)` ;
- `public Cellule(int valeur, Cellule suivante)`.

Construisez la liste donnée en exemple.

Pour tester le résultat, utilisez le code donné en annexe (Code 1) et vérifiez que le résultat est bien celui attendu.

Exercice 2 : Algorithmes itératifs

Structure de code

Voici un premier code de base pour afficher le contenu d'une liste :

```
public void affiche() {
  Cellule actuelle = this;
  while (actuelle != null) {
    System.out.println(actuelle.valeur);
    actuelle = actuelle.suivante;
  }
}
```

Elle se décompose ainsi :

- `Cellule actuelle = this;` : c'est la `Cellule` sur laquelle on est lors du parcours. Pourquoi vaut-elle `this` ?
- `while(actuelle != null)` : tant que je suis en train de travailler sur une `Cellule`, je vais faire une action ;
- `System.out.println(actuelle.valeur);` : je fais le travail sur ma `Cellule` actuelle ;
- `actuelle = actuelle.suivante` : lorsque j'ai terminé de travailler sur ma `Cellule` actuelle, je passe à la suivante (possiblement `null`).

> **Remarque**
>
> La même structure est utilisée dans Code 1.

Question 3 : Une petite somme
Faites une méthode `somme()` reprenant le même schéma pour faire la somme des éléments dans une liste.

Question 4 : Ajout au début
Ajoutez une méthode `Cellule ajoutDebut(int valeur)` qui renvoie une nouvelle liste où cette valeur a été ajoutée au début de `this` (pas besoin de copier la liste de départ).

Question 5 : Ajout à la fin
Ajoutez une méthode `void ajoutFin(int valeur)` qui ajoute la valeur `valeur` à la fin de la liste.

Question 6 : Ajout à la i-ème position
Ajoutez une méthode `void ajoute(int valeur, int position)` ajoutant la valeur `valeur` à la position `position`. On suppose que l'ajout est possible et la position strictement supérieure à 0.

Exercice 3 : Algorithmes récursifs

On a vu en cours le principe de récursion : faire le travail sur la partie actuelle puis passer au reste. Le but de cet exercice est d'appliquer ce principe aux listes chaînées.

Structure du code
Les premières fonctions récursives seront à peu près de la forme suivante :

```
public void afficheRec() {
  System.out.println(this.valeur);
  if (this.suivante != null)
    this.suivante.affiche();
}
```

Elle se décompose ainsi :

- `System.out.println(this.valeur);` : j'affiche la valeur de `this.valeur` ;
- `if (this.suivante != null)` : si je ne suis pas le dernier élément de la liste, `this.suivante` est différent de `null` ;
- `this.suivante.affiche()` : je demande à la `Cellule` suivante d'afficher la sous-liste dont elle est la première `Cellule`.

Question 7 : Comment ça fonctionne ?
Copiez ce code et testez-le sur différents exemples pour comprendre ce qu'il ce passe.

Question 8 : Affichage dans l'autre sens
Alors que la fonction donnée affichait 12 puis 99 et enfin 37, le but ici est de faire une méthode `afficheRecInverse()` qui affiche la liste dans l'ordre inverse (37 puis 99 puis 12).

Question 9 : Ajout à la fin
Ajoutez une méthode `ajouteFinRec(int valeur)` ajoutant une valeur à la fin de la liste.

Question 10 : Ajout à la i-ème position
Supposons que l'on cherche à ajouter une valeur donnée à la position i supposée strictement supérieure à 0 au départ et que l'ajout est possible.
Écrivez une telle méthode nommée `ajouteRec(int valeur, int position)` dans le style récursif.

Aide : Il y a deux cas possibles :

- i est égal à 1 : la valeur doit être la suivante
- i est strictement supérieur à 1 : c'est équivalent à ajouter à la $(i-1)$-ème position à partir de la `Cellule` suivante.

Partie 2 : Arbres binaires de recherche

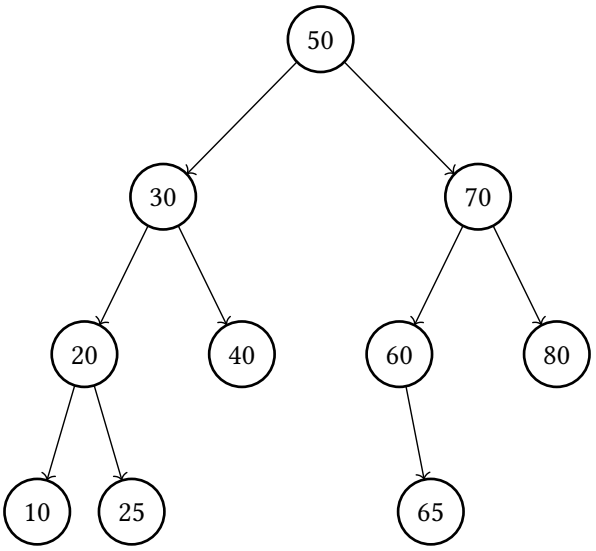

Fig. 2. – Structure d'un arbre binaire de recherche.

Exercice 4

Le but de cet exercice est d'implémenter une version simplifiée des arbres binaires de recherche.

Définition 1 (Arbre binaire de recherche). *Dans un arbre binaire de recherche chaque nœud porte une valeur et au plus deux fils (gauche et droit). De plus tous les nœud accessibles à partir du fils gauche ont une valeur inférieure à celle du nœud courant et tous les nœud accessibles à partir du fils gauche ont une valeur supérieure à celle du nœud courant.*

Même s'il est possible de le faire de manière itérative il est plus simple de le faire de manière récursive.

Question 1 : Construction de la classe

Modélisez un tel arbre avec une classe `Noeud`.

Question 2 : Constructeurs

Ajoutez un constructeur de `Noeud` qui prend une valeur et un autre qui prend une valeur et deux `Noeud` fils. Il n'y a pas besoin de vérifier que le résultat vérifie bien les propriétés des arbres binaires de recherche.

Question 3 : Ajout d'une valeur

Ajoutez une méthode `ajoute(int val)` qui ajoute la valeur la valeur `val` si elle n'est pas déjà présente dans l'arbre.
Même si ce n'est pas nécessaire vous pouvez utiliser la méthode de la question précédente.

Question 4 : Une petite somme

Ajoutez une méthode `somme()` calculant la somme des valeurs depuis le sous-arbre `this`.
Le résultat attendu pour l'arbre Fig. 2 est 450.

Question 5 : Est-ce un arbre binaire de recherche ?

Ajoutez une méthode `estBST()` qui vérifie si un arbre vérifie bien les propriétés d'un arbre de recherche (voir Définition 1).

Question 6 : Affichage
Ajoutez une méthode affiche `affiche()` qui va afficher la valeur du nœud courant, puis va faire de même sur le sous-arbre gauche puis sur le sous-arbre droit.
Pour l'arbre de la Fig. 2 le résultat est `50 30 20 10 25 40 70 60 65 80`.

Question 7 : Présence d'une valeur
Ajoutez une méthode `contient(int val)` indiquant si la valeur `val` est présente dans le sous-arbre représenté par `this`. Il faut éviter les sous-arbres où la valeur ne peut pas être présente.

Partie 3 : Arbres n-aires

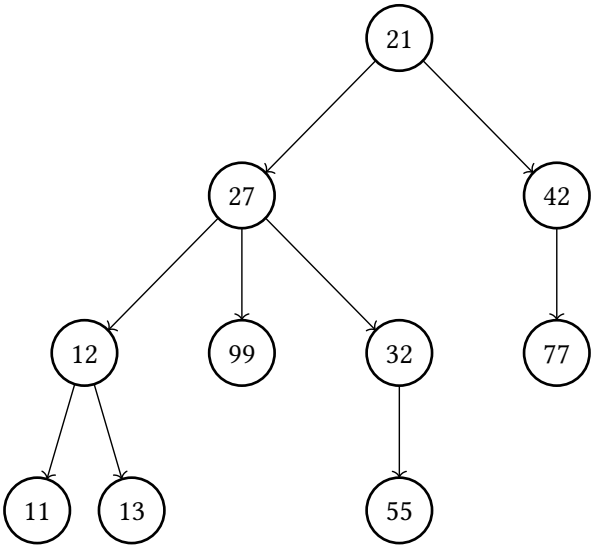

Fig. 3. – Structure d'un arbre n-aire.

Exercice 5

Définition 1 (Arbre n-aire). *Dans un arbre n-aire chaque nœud peut avoir un nombre quelconque de fils.*

Dans cet exercice sur chaque nœud il y aura un entier.

Question 1 : Construction de la classe
Modélisez un tel arbre avec une classe `NNoeud`. Vous pouvez utiliser la classe `ArrayList<NNoeud>` pour stocker les fils. Pour l'utiliser, vous devez ajouter au début de votre fichier la ligne suivante : `import java.util.ArrayList;`

Les seules méthodes qui devraient vous être utiles sont :

```
  ArrayList<NNoeud> liste = new ArrayList<NNoeud>(); // Création de la liste.
  NNoeud n = …;
  liste.add(n); // Ajoute à la fin
  liste.remove(0); // Supprime la valeur à la position donnée
  liste.contains(n); // La valeur est-elle présente ?
  NNode[] nodes = new Node[] {…};
  ArrayList<NNode> liste = new ArrayList<>(Arrays.asList(nodes)); // Création d'une
liste à partir d'un tableau.
```

Question 2 : Constructeurs
Créez un constructeur qui ne prend qu'un entier et un autre qui prend un entier et un tableau de fils.

Question 3 : Construction de l'exemple à la main
Construisez l'arbre de la figure Fig. 3 en n'utilisant que les constructeurs.

Question 4 : Une petite somme
Ajoutez une méthode `somme()` calculant la somme des valeurs depuis le sous-arbre `this`.
Le résultat attendu pour l'arbre Fig. 3 est 389.

Question 5 : Affichage
Ajoutez une méthode affiche `affiche()` qui va afficher la valeur du nœud courant, puis va faire de même sur le premier sous-arbre puis sur le deuxième etc.
Pour l'arbre de la Fig. 2 le résultat est `21 27 12 11 13 99 32 55 42 77`.

Question 6 : Présence d'une valeur
Ajoutez une méthode `contient(int val)` indiquant si la valeur `val` est présente dans le sous-arbre représenté par `this`. Il faut éviter les sous-arbres où la valeur ne peut pas être présente.

Annexe

ATTENTION : en fonction du lecteur de PDF il peut être impossible faire un copier-coller de ce code.

```
public void afficheVertical() {
  Cellule actuelle = this;

  final String format = "  +%s+\n  | %d |\n  +%s+\n";
  while (actuelle != null) {
    String line = "-".repeat(String.valueOf(actuelle.valeur).length() + 2);
    System.out.printf(format, line, actuelle.valeur, line);
    actuelle = actuelle.suivante;
    if (actuelle != null)
      System.out.println("    |   \n    v   ");
  }
  System.out.println("    |   \n    v   \n   null");
}
```

Code 1. – Méthode pour afficher une liste.