Séquence 17

Structure de donnée récursive

1 Introduction

La récursivité ne concerne pas seulement les traitements (les méthodes) mais également la représentation des données (classes). Dans une méthode récursive, il y a un appel à cette méthode. Dans une classe récursive, il y a une variable (ou plusieurs) dont le type est la classe elle-même.

2 Classe récursive

On appelle classe récursive une classe dans laquelle une ou plusieurs variables d'instance ont pour type le nom de la classe. C'est-à-dire qu'un objet instance de cette classe a des variables qui contiennent d'autres instances de la classe.

Prenons un premier exemple : celui d'une structure hiérarchique de type militaire où chaque individu a un supérieur hiérarchique unique. Nous allons définir la classe `Militaire` représentant un tel individu. Cette classe comprend un attribut de type `Militaire` pour contenir le supérieur de l'individu en question. Le supérieur d'un militaire est un militaire. Il a lui-même un supérieur.

```
class Militaire{
    String nom;
    String grade;
    Militaire superieur;
    Militaire(String n, String g, Militaire s){
        nom=n;
        grade=g;
        superieur=s;
    }
    public String getNom(){
        return nom;
    }
    public String getGrade(){
        return grade;
    }
    public Militaire getSuperieur(){
        return superieur;
    }
    public Militaire setSuperieur(Militaire sup){
        superieur = sup;
```

```
    }
    public String toString(){
        return nom + "␣" + grade;
    }
}
```

On voit que chaque objet de la classe Militaire contient une variable de type Militaire et que le constructeur permettant de créer un objet instance prend en paramètre un autre objet représentant son supérieur.

Deux questions se posent : comment créer le premier objet de la classe, alors qu'il n'existe aucun autre objet pour être le supérieur ; à la création de l'objet, comment new peut réserver une place en mémoire assez grande pour le militaire et son chef et le chef du chef, etc ?

La réponse à la première question, c'est qu'on peut créer un objet en donnant la valeur **null** pour représenter le supérieur. Cela permet de créer non seulement le premier objet, mais autant d'objets qu'on veut et qui n'ont pas de supérieur (à leur création).

La réponse à la seconde question, c'est que le new ne réserve de place en mémoire que pour le nouvel objet. Son supérieur (si ce n'est pas null) existe déjà dans la mémoire au moment de la création de l'objet. Il n'y a pas à lui réserver de place. De même pour le chef du chef.

Tout le secret de la structure récursive tient dans le mécanisme des références. La variable `superieur` ne contient pas directement le supérieur mais une référence, l'adresse en mémoire du supérieur. Cette adresse peut être null, pour représenter le cas où le militaire n'a pas de chef ou le cas où cette information sur son supérieur n'a pas encore été mise à jour.

Une structure récursive contient une ou plusieurs variables du même type qu'elle-même, comme une méthode récursive contient un ou plusieurs appels à elle-même.

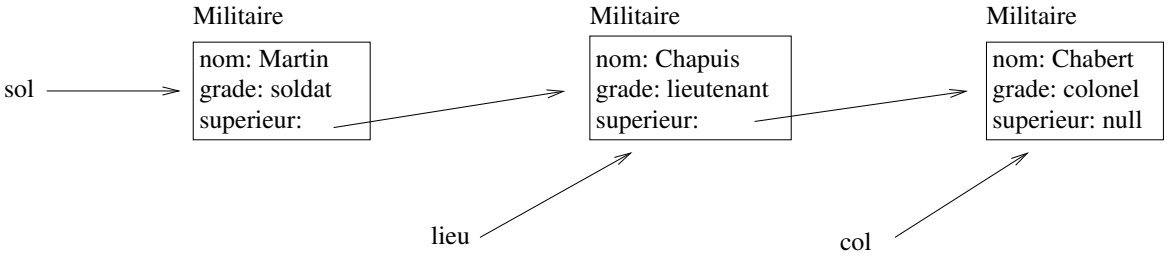
Autre chose : la méthode `toString` notée ici ne note pas dans la chaîne renvoyée le supérieur du militaire. Nous reviendrons là-dessus.

3 Construction d'une structure récursive

Voyons un programme qui utilise la classe `Militaire` pour construire effectivement une structure.

```
class TestMili{
    public static void main(String[] args){
        Militaire sol, lieu, col;
        col=new Militaire("Chabert","Colonel",null);
        lieu=new Militaire("Chapuis","lieutenant",col);
        sol=new Militaire("Martin","soldat",lieu);
    }
}
```

Ce pogramme construit une structure comportant trois objets de type `Militaire` reliés entre eux. On peut la représenter comme suit (pour simplifier le dessin, on ne sépare pas la pile du tas).

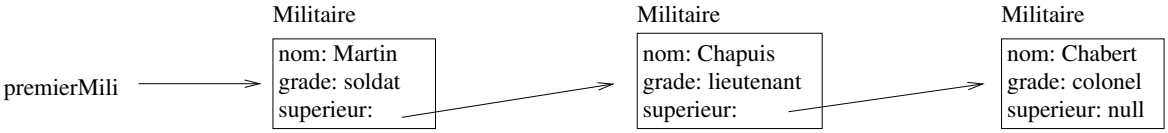


Dans ce premier exemple de structure, il y a une variable par objet créé. Mais généralement, ce n'est pas la cas, on a seulement une variable sur le premier élément de la structure et on peut atteindre les autres éléments via un chemin d'accès traversant plusieurs objets.

Voyons un exemple concret de programme et de structure où il y a une seule variable qui contient le premier objet de la structure.

```
class TestMili2{
    public static void main(String[] args){
        Militaire premierMili;
        premierMili=new Militaire("Chabert","Colonel",null);
        premierMili=new Militaire("Chapuis","lieutenant",premierMili);
        premierMili=new Militaire("Martin","soldat",premierMili);
    }
}
```

La structure construite peut se représenter comme suit.



Dans cette structure, on peut accéder au premier objet par le code `premierMili`, au second objet par `premierMili.getSuperieur()` et au troisième objet par le code `premierMili.getSuperieur().getSuperieur()`.

Une structure récursive contient indirectement, via des chaînes de références, plusieurs objets du même type. Beaucoup d'opérations courantes nécessitent de parcourir la structure pour accéder à certains ou à tous les objets de la structure. Ces opérations doivent fonctionner quel que soit le nombre d'objets dans la structure.

4 Parcours d'une structure récursive

La caractéristique du parcours dans la structure est qu'on ne peut pas accéder directement à chaque élément : il faut passer par des éléments intermédiaires. Par exemple, on ne peut pas accéder directement au chef du chef d'un soldat : on est obligé de passer par le chef du soldat et de regarder quel est son chef.

Le parcours d'une structure récursive peut se faire avec une boucle ou une méthode récursive, mais la version récursive est recommandable : elle est plus simple, plus naturelle.

Prenons comme premier exemple une méthode qui affiche toute la structure hiérarchique depuis le militaire courant (`this` pour la méthode) jusqu'à son supérieur le plus gradé.

Voici le code de deux méthodes récursives qui affichent la structure pour l'une du moins gradé au plus gradé, pour l'autre du plus gradé au moins gradé.

```
    public void afficheChaineHierachiqueCroissante(){
```

```
        System.out.println(this.toString());
        if (superieur != null){
            superieur.afficheChaineHierachiqueCroissante();
        }
    }
    public void afficheChaineHierachiqueDecroissante(){
        if (superieur != null){
            superieur.afficheChaineHierachiqueDecroissante();
        }
        System.out.println(this.toString());
    }
    public static void main(String[] args){
        Militaire premierMili;
        premierMili=new Militaire("Chabert","Colonel",null);
        premierMili=new Militaire("Chapuis","lieutenant",premierMili);
        premierMili=new Militaire("Martin","soldat",premierMili);
        premierMili.afficheChaineHierachiqueCroissante();
        System.out.println("=====================================");
        premierMili.afficheChaineHierachiqueDecroissante();
    }
```

L'éxécution du programme donne l'affichage suivant.

```
> java Militaire
Martin soldat
Chapuis lieutenant
Chabert Colonel
=====================================
Chabert Colonel
Chapuis lieutenant
Martin soldat
```

La différence entre les deux méthodes est seulement l'ordre entre le traitement de l'élément courant et l'invocation de la méthode récursive. Si l'on commence par l'invocation, c'est l'élément le plus lointain dans la structure (le colonel) qui est traité en premier. Si l'invocation est faite après, c'est le premier élément de la structure (le soldat) qui est traité en premier.

5 Méthode récursive qui renvoie un résultat

Notre premier exemple était une méthode d'affichage qui ne renvoie pas de résultat (type `void`). Voyons maintenant une méthode qui renvoie un résultat. Je vous propose une méthode qui recherche dans la structure un militaire par son nom. On va faire trois méthodes de recherches qui diffèrent par leur résultat : l'une aura un résultat booléen, oui ou non le militaire cherché est dans la chaîne hiérarchique ; la seconde renverra l'objet correspondant au militaire recherché et s'il n'est pas dans la structure, elle renvoie `null` ; la troisième méthode va calculer le rang du militaire dans la chaîne, en commençant à 0.

Dans les trois cas, la logique est identique :

— tester si le nom de l'objet courant est celui recherché.

— si oui, la méthode s'arrête, le résultat est trouvé.

— si non, si le supérieur vaut null, l'élément recherché n'est pas dans la structure.
— si le supérieur ne vaut pas null, invoquer récursivement la méthode sur le supérieur.

```
public boolean rechercheParNomBool(String achercher){
    if (nom.equals(achercher)){
            return true;
    }else{
        if (superieur == null){
            return false;
        }else{
            return superieur.rechercheParNomBool(achercher);
        }
    }
}
public Militaire rechercheParNomMili(String achercher){
    if (nom.equals(achercher)){
            return this;
    }else{
        if (superieur == null){
            return null;
        }else{
            return superieur.rechercheParNomMili(achercher);
        }
    }
}
public int rechercheParNomRang(String achercher){
    if (nom.equals(achercher)){
            return 0;
    }else{
        if (superieur == null){
            return -1;
        }else{
            int res = superieur.rechercheParNomRang(achercher);
            if (res==-1){
                return res;
            }else{
                return res+1;
            }
        }
    }
}
public static void main(String[] args){
    Militaire premierMili;
    premierMili=new Militaire("Chabert","Colonel",null);
    premierMili=new Militaire("Chapuis","lieutenant",premierMili);
    premierMili=new Militaire("Martin","soldat",premierMili);
    System.out.println("Chapuis␣dans␣la␣structure?␣" +
                        premierMili.rechercheParNomBool("Chapuis"));
    System.out.println("Chapuisat␣dans␣la␣structure?␣" +
                        premierMili.rechercheParNomBool("Chapuisat"));
    System.out.println("Chapuis␣dans␣la␣structure:␣" +
                        premierMili.rechercheParNomMili("Chapuis"));
    System.out.println("Chapuisat␣dans␣la␣structure:␣" +
```

```
                premierMili.rechercheParNomMili("Chapuisat"));
        System.out.println("Rang␣de␣Chapuis␣dans␣la␣structure:␣" +
                premierMili.rechercheParNomRang("Chapuis"));
        System.out.println("Rang␣de␣Martin␣dans␣la␣structure:␣" +
                premierMili.rechercheParNomRang("Martin"));
        System.out.println("Rang␣de␣Chabert␣dans␣la␣structure:␣" +
                premierMili.rechercheParNomRang("Chabert"));
        System.out.println("Rang␣de␣Martinet␣dans␣la␣structure:␣" +
                premierMili.rechercheParNomRang("Martinet"));
    }
```

Et voici la trace d'exécution :

```
> java Militaire
Chapuis dans la structure? true
Chapuisat dans la structure? false
Chapuis dans la structure: Chapuis lieutenant
Chapuisat dans la structure: null
Rang de Chapuis dans la structure: 1
Rang de Martin dans la structure: 0
Rang de Chabert dans la structure: 2
Rang de Martinet dans la structure: -1
```

6 Insertion dans une structure récursive

Dans les exemples vus jusqu'ici, la structure est construite grâce au constructeur. Cela permet de rajouter un nouvel élément en début de structure. Dans l'exemple, un constructeur ne peut ajouter qu'un moins gradé.

Il est possible d'insérer un nouvel objet au milieu de la structure en changeant le supérieur d'un objet existant. Voyons un exemple de code qui va insérer un capitaine entre le lieutenant et le colonel.

```
    public static void main(String[] args){
        Militaire premierMili, superieurDuCapitaine, capitaine;
        premierMili=new Militaire("Chabert","Colonel",null);
        premierMili=new Militaire("Chapuis","lieutenant",premierMili);
        premierMili=new Militaire("Martin","soldat",premierMili);
        superieurDuCapitaine = premierMili.getSuperieur().getSuperieur();
        capitaine = new Militaire("Dupont","capitaine",superieurDuCapitaine);
        premierMili.getSuperieur().setSuperieur(capitaine);
        premierMili.afficheChaineHierachiqueCroissante();
    }
```

Cette opération consistant à créer un nouvel objet et à changer le supérieur d'un objet existant peut être réalisé au moyen d'une méthode récursive. Celle-ci pourrait prendre en paramètre le rang où insérer le nouvel élément ou encore déduire l'emplacement d'une connaissance de l'ordre des grades : elle insérerait alors l'élément à la place qui convient pour que chaque supérieur ait un grade plus élevé que son inférieur.