Modélisation et Programmation par Objets 2

HAI401I

Classes imbriquées

Plan

Les classes imbriquées

Les classes imbriquées font partie des outils de modularisation, comme les paquetages, les classes ou les modules en Java 9.

Intérêts principaux :

- grouper des classes utilisées ensemble, souvent en identifiant une classe principale qui contient les autres.
- augmenter l'encapsulation :
  - suivant la visibilité choisie, la classe imbriquée peut être cachée.
- faciliter la communication et réduire des temps d'exécution :
  - sous certaines conditions, la classe imbriquée peut accéder à la partie privée de la classe englobante et inversement. Les accès sont plus directs et plus rapides.

Catégories

Les classes imbriquées se subdivisent en 2 principales catégories

- imbriquées statiques (*static nested classes*)
- imbriquées non statiques (3 sous-catégories) :
  - internes (*inner classes*)
  - locales (*local classes*)
  - anonymes (*anonymous classes*)

Plan

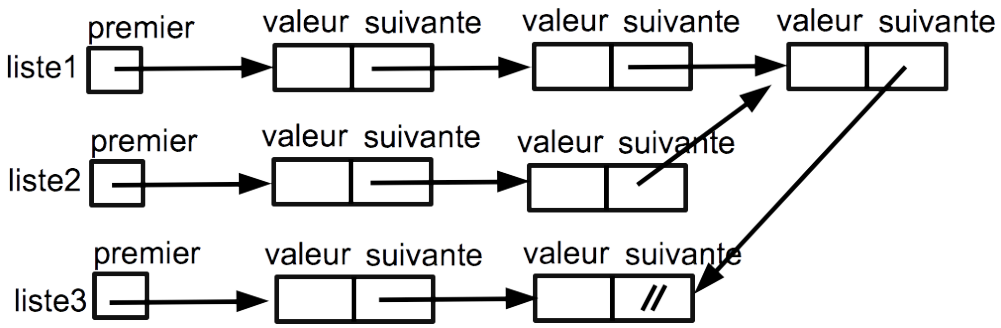
Classes imbriquées statiques

Principe des classes imbriquées statiques

- Moyen de ranger les classes les unes dans les autres pour des raisons thématiques.
- Pas de lien particulier entre une instance de la classe imbriquée et une instance de la classe englobante.
- À la visibilité près, la classe imbriquée pourrait être du même niveau d'imbrication que la classe englobante.

Classes imbriquées statiques

Liste chaînée avec partage des cellules entre listes (pas de lien privilégié entre liste et cellule)



Liste chaînée et cellule

```
public class Liste {
    private Cellule premier;
    private static int nbListes;
    (...)
    private static class Cellule {

      private int valeur;
      private Cellule suivante;
      (...)
      public String toString(){
        Liste l=new Liste();
        return ""+nbListes
        //+premier;
        +" "+l.premier;
      } // fin classe Cellule
```

`Liste` est la classe englobante
Elle a ses propres attributs, `premier`
et `nbListes` (ce dernier est static)

La class imbriquée `Cellule` est statique
Elle peut avoir différentes visibilités, ici `private`
`Cellule` a ses propres attributs `valeur`
et `suivante`

`Cellule` a sa propre méthode `toString`
On peut accéder aux membres static
de la classe englobante (comme `nbListes`).
Mais pas à `premier` qui n'est pas static
sauf sur une instance de `Liste`
créée explicitement

Liste chaînée et cellule

```
public class Liste {
.........
    public Liste() {}

    public void ajouteTete(int v){
        Liste.Cellule c = new Liste.Cellule();
        c.valeur=v;
        c.suivante=premier;
        premier=c;
        }

    public static void main(String[] a){
        Liste l = new Liste();
    }
} // fin classe Liste
```

`Liste` est la classe englobante

Constructeur de `Liste`

Méthode d'ajout dans une `Liste`
Création d'instance de la classe imbriquée
La classe englobante peut accéder
aux attributs privés de la classe imbriquée

Plan

Classe imbriquée non statique de type CLASSE INTERNE (*inner class*)

Classe imbriquée non statique de type CLASSE INTERNE

Exemple : structuration d'une classe `Personne`

Information sur une personne :

- nom, prénom,
- pays, ville, numéro de rue, nom de rue, code de postal,
- année de naissance.

Classe imbriquée non statique de type CLASSE INTERNE (*inner class*)

Classe imbriquée non statique de type CLASSE INTERNE

Exemple : structuration d'une classe `Personne`

Information sur une personne :

- nom, prénom,
- pays, ville, numéro de rue, nom de rue, code de postal (adresse)
- année de naissance.

La classe Adresse est une classe interne de la classe Personne

```
public class Personne {
  private String nom;
  private AdressePersonne ad;

  public Personne(String nom, String ville,
                  String pays) {
    this.nom = nom;
    ad = this.new AdressePersonne(ville, pays);
  }

  public String toString(){
    return nom+" habite "+ad;


    //return nom+" habite "+ad.ville;
  }
```

référence vers une instance de la classe interne

création d'une instance interne

code standard avec partage des responsabilités sur `toString` (on appelle `toString` sur `ad` plutôt que sur ses attributs)
exemple d'accès à un attribut de l'instance interne si nécessaire

La classe Adresse est une classe interne de la classe Personne

```
public class Personne {
.....    private class AdressePersonne{
      private String ville="là", pays="ailleurs";
      private static int nbAdresses=0;
      private static final String entete="Adresse";
      public AdressePersonne(String ville,
                             String pays)
      {this.ville = ville; this.pays = pays;}
      public String toString(){
        return ville+" "+pays;

        //return Personne.this.nom+this.ville+...;
        //return nom+this.ville+...;


        //return this.nom;
      }
   }

   public static void main(String[] args) {
      Personne j =
           new Personne ("Jacky","Boston","USA");
      System.out.println(j);
   }
}
```

on ne peut pas avoir une variable `static`
on peut avoir une variable `static` ET `final`

code standard avec partage des responsabilités sur `toString`
accès à un attribut de l'instance englobante
accès à un attribut de l'instance englobante (alternative montrant que l'instance englobante est implicite)
mais `this.nom` provoque une erreur !

Plan

Classes imbriquée non statique de type classe locale (*local class*)

Classe imbriquée locale

- classe interne créée dans un bloc (méthode, corps d'une itération, etc.)
- portée sera restreinte au bloc
- structure le bloc en lui ajoutant des déclarations de types qui ne sont pas utiles en d'autres endroits du programme.
- permet d'expliciter de la connaissance métier enfouie dans le code.

Classes imbriquée non statique de type classe locale (*local class*)

Code de saisie des éléments d'un numéro de téléphone dans un flot, qui retourne une chaîne construite avec un numéro utilisable sur un téléphone :

```
...
public String saisieNumeroTelephone(Scanner sc){

System.out.println("indicatif ?");
String i = sc.next();
System.out.println("numéro local ?");
String j = sc.next();
return "+"+i+j;
}
```

Classes imbriquée non statique de type classe locale (*local class*)

information du domaine non explicitée :

- concept de "Numéro de Téléphone" (pourrait être une classe `NumTel`) ;
- objets de `NumTel` décrits par :
  - un indicatif (pour simplifier, spécifique à chaque numéro de téléphone) ;
  - un numéro local (information spécifique à chaque numéro de téléphone) ;
  - un préfixe (ou code) "+" partagé, global à tous les numéros de téléphone.

```
...
public String saisieNumeroTelephone(Scanner sc){

System.out.println("indicatif ?");
String i = sc.next();
System.out.println("numéro local ?");
String j = sc.next();
return "+"+i+j;
}
```

Classe locale

```
public class Autorisation {
   private int num;
   public Autorisation() {}

   public String saisieNumTel(Scanner sc) {
      System.out.println("saisie num tel");
      String numComplet;
      String code="+";

      class NumTel{
      private String indicatif;
      private String numerolocal;

      public String saisie(Scanner sc){
         System.out.println("saisie indicatif");
         indicatif = sc.next();
         System.out.println("saisie numero");
         numerolocal = sc.next();
         //numComplet = indicatif+" "+...;
         return code+indicatif+" "+numerolocal;
            //+Autorisation.this.num;
            //+num;
         }
       } // fin classe NumTel
```

On met la classe dans la méthode saisieNumTel
pas de visibilité, pourrait être abstraite ou final

on ne peut affecter une valeur à numComplet ici
on peut utiliser la variable locale code
accès possible à l'instance implicite
idem (comme dans une classe interne)

Classe locale

```
public class Autorisation {
....
        NumTel n = new NumTel() ;
        numComplet = n.saisie(sc) ;
        return numComplet ;
    }//fin saisieNumTel

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in) ;
        Autorisation a = new Autorisation() ;
        System.out.println(a.saisieNumTel(sc)) ;
    }
}
```

suite de la méthode saisieNumTel
création d'un objet de la classe locale

Plan

Classe imbriquée non statique de type classe anonyme (*anonymous class*)

Classe anonyme :

- variante des classes locales
- ne possède pas de nom : uniquement un corps de classe
- construite sur la base d'une classe support (ou d'un type interface) préalablement déclarée, et qui peut être abstraite
- très utilisée en programmation graphique pour définir des objets temporaires (bouton, menu)

Classe anonyme

```
public class Saisissable{
   public String saisie(Scanner sc){
      System.out.println("On peut me saisir !");
      return ""; } }

public class AutorisationAnonyme {
   public String saisieNumTel(Scanner sc) {
      System.out.println("saisie num tel");
      String code="+";
      Saisissable n = new Saisissable(){
         private String indicatif;
         private String numerolocal;
         public String saisie(Scanner sc){
            System.out.println("saisie indicatif");
            indicatif = sc.next();
            System.out.println("saisie numero");
            numerolocal = sc.next();
            return code+indicatif+" "+numerolocal;
         }
      }; // fin classe anonyme
      return n.saisie(sc);
   }//fin saisieNumTels
// le main est inchangé
... }
```

définition de la classe support
mais ici la classe `Object` aurait pu servir
de support également (voir listing suivant)

définition de la classe anonyme
par spécialisation de `Saisissable`
et instanciation en même temps

Classe anonyme

Variante avec utilisation de la classe `Object` comme base

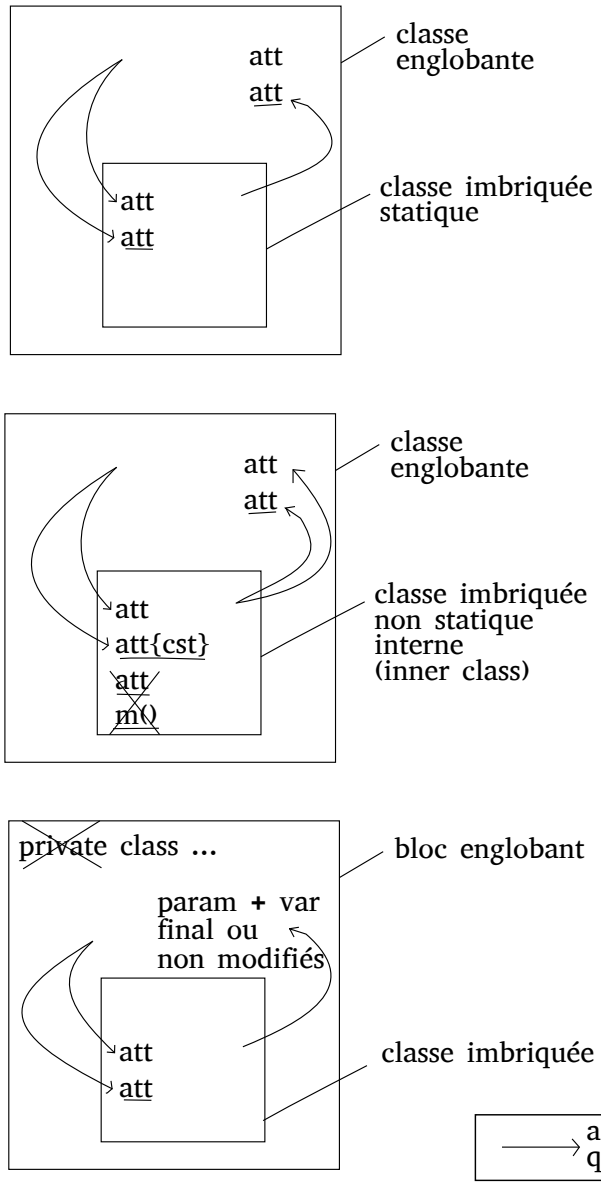
```
public class AutorisationAnonyme {

   public String saisieNumTel(Scanner sc) {
      System.out.println("saisie num tel");
      return new Object() {
        private String indicatif;
        private String numerolocal;
        private final static String code="+";
        public String saisie(Scanner sc){
         System.out.println("saisie indicatif");
         indicatif = sc.next();
         System.out.println("saisie numero");
         numerolocal = sc.next();
         return code+indicatif+" "+numerolocal;
        }
      }.saisie(sc);
   }//fin saisieNumTel
.... }
```

classe anonyme

Appel de la saisie sur l'objet directement

suite (main) identique

Vue synthétique des classes imbriquées



Modules (Java 9)

Ouverture vers les modules Java 9

Groupe nommé de paquetages et de ressources, muni d'un descripteur contenant

- nom du module
- dépendances (autres modules)
- paquetages rendus disponibles depuis d'autres modules
- services fournis (offerts)
- services requis (consommés)

Modules (Java 9)

Objectifs

- Inspiré par le paradigme du développement orienté composants
- Facilite la configuration (explicitation des dépendances)
- Renforcement de l'encapsulation (explicitation des exports permis)
- Optimisation des performances, introduction d'une phase d'édition de liens (entre la compilation et l'exécution)
- Modularisation de la plateforme Java elle-même ($\sim$100 modules) qui peut causer des problèmes de migration (certaines APIs sont devenues internes)

Pour aller plus loin `https://www.baeldung.com/java-9-modularity`

Synthèse

Modularité en Java par niveau descendant d'imbrication :

- Modules en Java 9
- Packages
- Classes
- Classes imbriquées