

Introduction à la Programmation : Classes imbriquées

Laboratoire d'Intelligence Artificielle
Faculté I&C

Objectifs du cours de la semaine

- ▶ Les classes imbriquées
- ▶ Présentation du mini-projet 2

Classes imbriquées : introduction

Supposons que nous souhaitions modéliser une grille sur laquelle se déroule un jeu (jeu d'échec, RPG etc.)

- ▶ Il s'agit de modéliser la grille comme un ensemble de cellules
👉 type `Grid`
- ▶ Les cellules sont de objets à part entière (peuvent avoir un contenu, des attributs etc.) 👉 type `Cell`
- ▶ La notion de cellule est solidaire de celle de grille
👉 il s'agit de *cellules d'une grille*

Classes imbriquées : introduction

Modéliser séparément la notion de grille et la notion de cellule n'est pas très adapté :

- ▶ ces deux entités doivent pouvoir communiquer entre elles sans restriction
- ▶ les séparer implique de coder une API qui assure cette communication (par exemple, `Cell` devrait fournir un accesseur `getContent()` s'il est nécessaire que son contenu soit connu de la grille).

👉 Solution : **imbriquer** la classe `Cell` dans la classe `Grid`

Classes imbriquées : syntaxe

Pour imbriquer une classe dans une autre, il suffit d'écrire :

```
public class OuterClass
{
    private class InnerClass
    {
        /* declaration and definition
           of the members of InnerClass
        */
    }
    /* declaration and definition
       of the members of OuterClass
    */
}
```

Droit d'accès privé de la classe interne

👉 hypothèse qu'elle n'est utile qu'à la classe externe

Terminologie : classes imbriquées = «*nested classes*»

Classes imbriquées : droit d'accès (1)

Du point de vue de la **compilation** :

- ➡ les classes internes et externe **peuvent accéder mutuellement à tous leurs membres** quelqu'en soit les droits d'accès.

Classes imbriquées : droit d'accès (2)

Exemple :

```
public class Grid
{
    private Cell[][] cells;
    public Grid(int lines, int columns)
    {
        cells = new Cell[lines][columns];
        for (int i=0; i<lines; ++i)
            for (int j=0; j<columns; ++j) {
                cells[i][j] = new Cell(i,j);
            }
    }
    private class Cell
    {
        private int line;
        private int column;
        private Cell(int i, int j)
        {
            assert (cells != null && i >=0 && j>=0);
            // other useful assertions
            line = i;
            column = j;
        }
    }
}
```

Classes imbriquées : droit d'accès (3)

Du point de vue de l'**exécution** :

- ▶ Une classe interne peut accéder inconditionnellement à toutes les méthodes de sa classe externe
- ▶ Une classe externe ne peut accéder sans erreur à une méthode de la classe interne que si des objets de cette classe ont été créés !

Classes imbriquées : droit d'accès (4)

Par exemple, si l'on code une méthode `display` dans `Cell` :

```
private void display()
{
    System.out.println("Cell " + line + " " + column);
}
```

L'invoquer dans `Grid`, implique que des objets de type `Cell` aient été préalablement créés :

```
public void display()
{
    for (int i=0; i<cells.length; ++i)
        for (int j=0; j<cells[0].length; ++j) {
            cells[i][j].display();
        }
}
```

👉 dans notre exemple, c'est fait par le constructeur de `Grid`

Classes imbriquées : accès aux méthodes

Lorsqu'une méthode de la classe interne appelle une autre méthode `m` :

- ▶ `m` est cherchée dans la classe interne et si elle n'est pas trouvée elle est cherchée dans la classe externe
- ▶ si `m` existe dans les deux classes, une méthode de la classe interne peut invoquer la méthode `m` de la classe externe au travers de l'instance courante de cette dernière
- ▶ l'instance courante de la classe externe est accessible selon la syntaxe suivante : `OuterClass.this`

👉 exemple sur la page suivante ...

Classes imbriquées : Accès aux méthodes(2)

```
class Grid
{
    private String name;
    //... comme avant
    public void display() {
        //..
    }
    public void displayGridName() {
        System.out.println(name);
    }
    private class Cell
    {
        // ... comme avant
        private void display() {
            System.out.println("Cell " + line + " " + column);
        }
        private void displayOwnerGrid() {
            displayGridName();
            Grid.this.display();
        }
    }
}
```

Classes imbriquées statiques

```
class Grid
{
    private static class Tile
    {
        private Orientation orientation;
        private final Sprite upSprite;
        private final Sprite rightSprite;
        private final Sprite downSprite;
        private final Sprite leftSprite;
        private Sprite currentSprite;

        public Tile(...)
        {
            //...
        }
        public Sprite getSprite() {}
    }
}
```

- 📌 Lorsque la classe interne n'a pas besoin d'accéder à l'instance courante de la classe externe, elle peut être déclarée comme statique.

Classes imbriquées statiques (2)

Une classe interne statique :

- ▶ Peut avoir des membres aussi bien statique que non statiques
- ▶ Elle ne peut accéder à aucun membre non statique de la classe externe
- ▶ (Si la classe interne a des membres statiques alors elle doit être statique) : cette contrainte a été levée depuis Java 16.

Terminologie : pour les différencier des classes imbriquées statiques, les classes imbriquées non statiques sont souvent désignées sous le vocable de **classes internes** («*inner classes*»),

Classes imbriquées publiques (1)

Revenons à notre exemple de grille de jeu. Supposons qu'un jeu soit représenté par une grille et un ensemble d'acteurs qui y prennent place.

Supposons qu'il soit nécessaire de définir une méthode d'interaction entre un acteur du jeu et une cellule

- ➡ Le type `Cell` doit devenir accessible en dehors de la classe `Grid`
- ➡ On peut en faire une classe imbriquée **publique**.

Classes imbriquées publiques (2)

- ▶ Une classe imbriquée publique peut être utilisée de toutes les façon déjà décrites pour le cas privé
- ▶ L'accès au type offert par la classe interne se fait par la notation `OuterClass.InnerClass`

Exemple : `void interact(Actor a, Grid.Cell cell)`

- ☞ Cela pose le problème de la construction d'objet du type de la classe interne en dehors de la classe externe
- ☞ Traité au second semestre

Classes imbriquées et héritage (2)

- ▶ Si une classe `OuterClass` contient une classe imbriquée `InnerClass` alors toute classe héritant de `OuterClass` hérite aussi de la classe imbriquée `InnerClass`
- ▶ La classe interne peut elle même hériter de n'importe quelle classe accessible
- ▶ Il n'est cependant pas possible de redéfinir une classe imbriquée dans une sous-classe