UEF4.3.Programmation Orientée Objet

Mme SADEG
s_sadeg@esi.dz
Ecole nationale Supérieure d'Informatique (ESI)
2011/2012

Chapitre IV

Classes internes et paquetages

1. Classes internes

Classe interne

- Une classe est dite interne lorsque sa définition est située à l'intérieur d'une autre classe.
- Les classes internes ont essentiellement pour but de simplifier l'écriture du code de la programmation évènementielle (dans la programmation graphique)
- Il y a plusieurs types de classes internes:
  - Les classes internes simples (non statiques)
  - Les classes internes statiques
  - Les classes locales
  - Les classes anonymes

Classe interne non statique

- Elle est définie à l'intérieur d'une autre classe (appelée principale, englobante ou externe) en tant que membre de cette dernière.
- Elle a l'avantage de pouvoir accéder aux autres membres de la classes même s'il sont privés

```
class E                 //définition d'une classe usuelle dite
                        //externe, principale ou englobante
{......                 //méthodes et données de la classe E
  class I               //Définition d'une classe interne à la
                              classe E
  {….                   // méthodes et données de la classe I
  }
…..                     // autres méthodes et données de la
                              classe E
}
```

Utilisation d'une classe interne

- La définition d'une classe interne I est utilisable au sein de la classe principale E.

Exemple 1

```
class E
{public void fe()      // méthode de E
{ I i = new I();       //Création d'un objet de type I; Sa
                       //référence est locale à la méthode fe
}
  class I
  {….
  }
…..
}
```

- Ici, un objet de type E ne contient aucun membre de type I. Simplement, une de ses méthodes utilise un objet de type I

Utilisation d'une classe interne

- On peut aussi trouver une ou plusieurs références à des objets de type I au sein de la classe E comme dans cet exemple:

Exemple 2

```
class E
{
private I i1, i2;    // les champs i1 et i2 sont des
                     //références à des objets de type I

  class I
  {….
  }
…..
}
```

Lien entre objet interne et objet externe

- En quoi la définition d'une classe I interne à une classe E diffère-t-elle d'une définition d'une classe I externe à la classe E ?
- Les objets correspondant à cette situation de classe interne jouissent de trois particularités
  1. Un objet de la classe I est associé au moment de sa création à un objet de type E dont on dit qu'il lui a donné naissance. Dans l'exemple 1, l'objet de type I sera associé à l'objet de type E auquel sera appliquée la méthode *fe*.
  2. Un objet d'une classe interne a toujours accès aux champs et méthodes (même privés) de l'objet de type E qui lui a donné naissance (mais pas aux autres objets de E)
  3. Un objet de type E a toujours accès aux champs et méthodes (même privés) d'un objet de type I auquel il a donné naissance

Remarques:

- Une méthode statique d'une classe externe ne peut créer aucun objet d'une classe interne
- Une classe interne ne peut contenir de membres statiques

Déclaration et création d'un objet d'une classe interne

- En théorie (mais c'est rarement exploité), java permet d'utiliser une classe interne depuis une classe indépendante (non englobante). Mais il faut quand même rattacher un objet d'une classe interne à un objet de sa classe englobante, en utilisant une syntaxe particulière de new.

Exemple

```
class E
{......
   class I
   {….
   }
…..
}
```

En dehors de E, on peut t déclarer une référence à un objet de type I, Mais la création d'un objet de type I ne peut se faire qu'en le rattachant à un objet de type E :

```
E e; E.I i;
E e = new E();
i = e.new I();
```

Classes internes statiques

- Les objets des classes internes vues jusqu'à maintenant étaient toujours associés à un objet d'une classe englobante. On peut créer une classe interne « autonome » en utilisant le mot clé static:

```
class E                          // classe englobante
{......                          //méthodes et données de la
                                 //classe E
  public static class I          //Définition dans la classe E
                                 //d'une classe interne autonome
  {….
  }
…..
}
```

Classes internes statiques

- A l'extérieur de la classe E, on peut instancier un objet de la classe I de cette façon:

```
E.I i = new E.I();
```

- L'objet i n'est associé à aucun objet de type E. La classe I n'as accès à aucun membre de E, sauf s'il s'agit de membre statiques.

Classes locales

- On peut définir une classe interne I dans une méthode f d'une classe E. dans ce cas l'instanciation d'objets de type I ne peut se faire que dans *f*. En plus des accès déjà décrits, un objet de type *I* a alors accès aux variables finales de *f*.

```
Public class E
{……
void f()
{
I i = new I();
final int n = 15;
float x;

class I          // c'est une classe interne à la classe E et
                //locale à la méthode f. ici on a accès à n mais
                //pas à x
{…..
}

}
……
}
```

Classes anonymes

- Les classes internes anonymes (anonymous inner-classes) sont des classes internes qui ne possèdent pas de nom. Elles ne peuvent donc être instanciées qu'à l'endroit ou elles sont définies.
- Ce type de classe est très pratique dans la gestion des événements ou lorsqu'une classe doit être utilisée une seule fois
- La syntaxe de définition d'une classe anonyme ne s'applique que dans deux cas:
  - Classe anonyme dérivée d'une autre
  - Classe anonyme implémentant une interface

Classes anonymes

Classe anonyme dérivée d'une autre classe

```
class A {
  public void afficher() {
      System.out.println("je suis un A ");
  }
}

public class ExempleAnonyme1 {
  public static void main(String args[]) {
   A a;
   a = new A() { //début de la classe anonyme
       public void afficher() {
       System.out.println(" je suis une classe anonyme dérivée de A");
       }
     }; // fin: le point-virgule est obligatoire

   a.afficher(); //appel la méthode afficher de la classe anonyme
  }
}
```

Classes anonymes

Classe anonyme implémentant une interface

```
interface Affichable {
   public void afficher();
}

public class exempleAnonyme2 {
   public static void main(String args[]) {
   Affichable a;
   a = new Affichable() {//début de la classe anonyme
       public void afficher() {
         System.out.println("je suis une classe anonyme implémentant
                                 Affichable ");
         }
   }; // le point-virgule est obligatoire

   a.afficher(); // Appel afficher de la classe anonyme
  }
}
```

1. Paquetages

Les paquetages

- Un paquetage est un ensemble de classes regroupées sous le même identificateur. Cette notion est proche de celle de bibliothèque dans d'autres langages.

- L'organisation des classes en paquetages facilite le développement de logiciels de taille importante car les conflits de noms sont limités aux classes appartenant au même paquetage.

Attribution d'une classe à un paquetage

- Un paquetage est caractérisé soit par un simple identificateur, soit par une suite d'identificateurs séparés par des points (dits « paquetages hiérarchisés »)

```
MesClasses
Utilitaires.Mathématiques
Utilitaires.Tris
```

- L'attribution d'un nom de paquetage se fait au niveau du fichier source. Toutes les classes appartenant au même fichier source appartiennent au même paquetage. Cela se fait en plaçant au début du fichier l'instruction:

```
package xxx; //xxx étant le nom du paquetage
```

La notion de paquetage est logique

- La notion de paquetage est « logique » et n'a pas de rapport avec l'emplacement effectif des classes ou des fichiers au sein des répertoires.

- Quand on utilise des noms de paquetage hiérarchisés, il ne s'agit que d'une organisation logique des noms de paquetages. En effet pour désigner simultanément `utilitaires.Mathématiques` et `utilitaires.Tris`, on ne peut pas se contenter de citer `Utilitaires`.

- Si aucun nom de paquetage n'est spécifié (pas d'instruction `package`), le compilateur considère que les classes correspondantes appartiennent au paquetage par défaut.

Utilisation d'une classe d'un paquetage

- Lorsqu'une classe est référencée dans un programme, le compilateur la recherche dans le paquetage par défaut. Pour utiliser une classe appartenant à un autre paquetage, il faut « informer » le compilateur et pour cela on peut :
  1. Citer le nom du paquetage avec le nom de la classe
  2. Utiliser l'instruction `import` en citant une classe particulière d'un paquetage,
  3. Utiliser l'instruction import en citant tout un paquetage

Utilisation d'une classe d'un paquetage

1. En citant le nom de la classe

- Si par exemple on attribue à la classe Point le nom de paquetage MesClasses, on peut l'utiliser simplement en la nommant Mesclasses.Point:

```
MesClasses.Point p= new MesClasses.Point(3,5);
….
p.affiche();
```

Cette démarche peut s'avérer fastidieuse si de nombreuse classes sont concernées

Utilisation d'une classe d'un paquetage

2. En important une classe

L'instruction **import** permet d'importer le nom d'une ou plusieurs classes. Le nom doit être complet.

```
import MesClasses.Point, MesClasses.Cercle;
```

A partir de là, les classes *Point* et *Cercle* pourront être utilisées sans avoir à mentionner le nom du paquetage.

Si un certain nombre de classes du même paquetage sont concernées, la troisième possibilité est préférable.

Utilisation d'une classe d'un paquetage

3. En important un paquetage

Pour utiliser toutes les classes d'un même paquetage, il suffit d'importer le paquetage entier:

```
import MesClasses.*;
```

Attention, si on a créé un paquetage appelé MesClasses.TP, ses classes ne seront pas concernées par cette instruction.

Utilisation d'une classe d'un paquetage

Remarques

- Si deux paquetages contenant des classes de même nom sont importés, cela constitue une erreur.
- Citer un paquetage sans utiliser toutes ses classes n'augmente ni le temps de compilation ni la taille des fichiers .class générés. L'utilité du filtrage est de limiter les ambiguïtés en cas de classes de noms similaires entre différents packages importés.
- La plupart des environnement imposent des contraintes concernant l'emplacement des fichiers correspondant à un paquetage.
  - Un paquetage de nom X.Y.Z se trouve toujours entièrement dans un sous-repertoire appelé X.Y.Z.
  - Mais le paquetage X.Y.U qui se trouve dans le sous répertoire X.Y.U peut se trouver dans un répertoire différent de celui de X.Y.Z.
  - Avec le SDK (ou JDK), la recherche d'un paquetage se fait dans les répertoires déclarés dans la variable d'environnement CLASSPATH

Les paquetages standard

- Les classes standard fournies avec java sont structurées en paquetages comme par exemple java.awt, java.awt.event,javax.swing…

- Il existe un paquetage particulier nommé java.lang qui est automatiquement importé par le compilateur et qui permet d'utiliser les classes standard telles que Math, System, Float ou Integer, Double, Object…

Paquetages et droits d'accès

Droits d'accès aux classes

Chaque classe dispose d'un droit d'accès (ou modificateur d'accès) qui permet de décider quelles sont les autres classes qui peuvent les utiliser. Le droit d'accès est précisé par la présence ou l'absence du mot clé public:

- avec le mot clé public, la classe est accessible à toutes les autres classes (En ayant éventuellement recours à une instruction import)
- Sans le mot clé public, la classe n'est accessible qu'au classes du même paquetage

Tant que l'on travaille avec le paquetage par défaut, le mot clé public est facultatif sauf pour la classe contenant la méthode main qui doit être publique pour que la JVM puisse y avoir accès

Paquetages et droits d'accès

Droits d'accès aux membres d'une classe

Nous avons déjà vu qu'on pouvait utiliser pour un membre (champs ou méthode) l'un des modificateurs d'accès

- *public* (le membre est accessible de l'extérieur de la classe),
- *private* (le membre n'est accessible qu'aux méthodes de la classe) et
- *protected* (le membre est accessible par les classes dérivées).

Il existe une quatrième possibilité qui est l'absence de modificateurs d'accès qui limite l'accès aux classes du même paquetage. Ce droit d'accès est appelé *accès paquetage*.

```
package P1;
public class A // accessible partout
{...
void f1() {....}
public void f2(); {...}
}
```

```
package P2
class  B// accessible uniquement dans p2
{...
public void g()
{ A a = new A();
a.f1();       // interdit
a.f2();      // OK
}
}
```

Classes internes et droits d'accès

Les classes internes sont concernées par les droits d'accès mais leur signification est différente et se rapproche de la définition des droits d'accès aux membres:

- Une classe interne déclarée publique est accessible partout où sa classe externe l'est.
- Une classe interne déclarée privée (ce qui n'est pas utilisable pour une classe externe) n'est accessible que depuis sa classe externe
- Sans aucun mot clé, la classe interne n'est accessible que depuis les classes du même paquetage.

Modificateurs d'accès des classes et interfaces

Modificateur	Signification pour une classe ou une interface
public	Accès toujours possible
néant (droit de paquetage)	Accès possible depuis les classes du même paquetage

Modificateurs d'accès pour les membres et les classes internes

Modificateur	Signification pour un membre d'une classe (champs ou méthode) ou une classe interne
public	Accès possible partout où sa classe est accessible
néant (droit de paquetage)	Accès possible depuis toutes les classes du même paquetage (quelque soit le droit d'accès de la classe qui est au minimum celui du paquetage)
protected	Accès possible depuis toutes les classes du même paquetage ou depuis les classes dérivées
private	Accès restreint à la classe où est faite la déclaration du membre ou de la classe interne