Classes imbriquées en Java

B. Mermet
2010

Types de classes imbriquées

- Classes statiques
- Classes non statiques (= classes *internes*)
    - Standards
    - Locales
        - Locales
        - Anonymes

Classes imbriquées statiques

- Déclaration
  - Dans une classe
  - *Visibilité* `static class NomClasse {`

    ...

    `}`
- Visibilité : public, protégée, package, privée
- Utilisation : ClasseEnglobante.ClasseImbriquée
- Particularité : accès aux variables de classe de la classe englobantes, mêmes privées
- Intérêts
  - structuration plus fine des classes

Classes imbriquées statiques Exemple

```
public class ClasseEnglobanteImbriqueeStatique {

    private int x;
    private static int n = 0;

    public ClasseEnglobanteImbriqueeStatique() {
        x=n++;
    }

    public static class MaClasseImbriqueeStatique {
        public MaClasseImbriqueeStatique() {
            System.out.println("n = "+n);
        }
    }

    public static void main(String[] args) {
        MaClasseImbriqueeStatique m = new MaClasseImbriqueeStatique();
        ClasseEnglobanteImbriqueeStatique c = new ClasseEnglobanteImbriqueeStatique();
        m = new ClasseEnglobanteImbriqueeStatique.MaClasseImbriqueeStatique();
    }
}
```

Classes imbriquées statiques Commentaires sur l'exemple

- Noter l'accès à "n" dans la classe imbriquée, avec n variable statique privée de la classe englobante
- Dans le main, la première création d'une instance de la classe imbriquée semble classique. En effet, on est déjà dans la classe englobante
- La deuxième création utilise la syntaxe à utiliser dans toute autre classe
- L'affichage produite est donc :
    - 0
    - 1

Classes internes

- Déclaration
  - Dans une classe
  - *Visibilité* `class NomClasse{...}`
- Visibilité : public, protégée, package, privée
- Utilisation
  - Création : InstanceClasseEnglobante.new ClasseInterne(...)
  - Type : ClasseEnglobante.ClasseInterne
- Particularité
  - accès aux variables d'instance de la classe englobantes, mêmes privées
  - Accès à l'objet englobant associé : ClasseEnglobante.this
- Intérêts
  - structuration plus fine des classes
  - Implantation de la notion de "classe amie"

Classes internes
Exemple

```
public class ClasseEnglobanteInterne {
    private int x; private int y; private boolean b;
    public ClasseEnglobanteInterne(int x, int y) {this.x = x; this.y = y; b = true;}
    public String toString() {return "("+x+","+y+","+b+")";}
    public class ClasseInterne {
        private int z; private boolean b;
        public ClasseInterne(int z) {this.z = z; b = false;}
        public String toString() {
            return ClasseEnglobanteInterne.this.toString()+"-"+"("+x+","+y+","+z+","+b+")";}
    }

    public static void main(String[] args) {
        ClasseEnglobanteInterne ce1 = new ClasseEnglobanteInterne(3,4);
        ClasseEnglobanteInterne ce2 = new ClasseEnglobanteInterne(13,14);
        ClasseInterne ci1a = ce1.new ClasseInterne(5);
        ClasseInterne ci1b = ce1.new ClasseInterne(6);
        ClasseInterne ci2 = ce2.new ClasseInterne(15);
        System.out.println(ce1); System.out.println(ce2);
        System.out.println(ci1a); System.out.println(ci1b);
        System.out.println(ci2);
    }
}
```

Classes internes
Commentaires sur l'exemple

- Les variables d'instances x et y de la classe englobante peuvent être utilisées sans problème dans la classe interne
- La variable b de la classe interne masque, dans celle-ci, la variable b de la classe englobante
- Dans le toString() de la classe interne, noter l'utilisation de ClasseEnglobanteInterne.this.toString() pour appeler la méthode toString() de la classe englobante. En effet, si on avait écrit directement "toString()", on aurait alors appelé le toString() de la classe interne...
- Dans le main, remarquer la création des instances de la classe interne.
- Noter que plusieurs instances de la classe interne peuvent être associées à une même instance de la classe englobante.
- L'affichage produit est donc :
    - (3,4,true)
    - (13,14,true)
    - (3,4,true)-(3,4,5,false)
    - (3,4,true)-(3,4,6,false)
    - (13,14,true)-(13,14,15,false)

Classes internes locales

- Déclaration
  - Dans un bloc de code (constructeur, méthode, ...)
- Visibilité : hors de propos
- Utilisation
  - Type visible uniquement dans le bloc de code englobant
  - Si instance retournée par la méthode, utiliser un super-type (super-classe, interface implantée) comme type de retour
- Particularité
  - accès aux variables d'instance de la classe englobantes, mêmes privées
  - Accès à l'objet englobant associé : ClasseEnglobante.this
  - Accès aux paramètres et variables locales du bloc englobant s'ils sont "final"
- Intérêt
  - Retourner des instances de classes différentes suivant les cas
  - Retourner des instances paramétrées par l'appel à la méthode englobante

Classes internes locales Exemple

```
public class ClasseEnglobanteLocale {
    private int x;
    public ClasseEnglobanteLocale(int x) {this.x = x;}
    Object setX(final int v) {
        x = v;
        final int tmp = new Random().nextInt();
        class ClasseLocale {
            private int n;
            public ClasseLocale(int w) {n = w;}
            public String toString() {return ""+n+";"+v+";"+tmp;}
        }
        return new ClasseLocale(v+10);
    }
    public static void main(String[] args) {
        ClasseEnglobanteLocale cel = new ClasseEnglobanteLocale(2);
        Object o1 = cel.setX(3);Object o2 = cel.setX(4);
        System.out.println(o1); System.out.println(o2);
        System.out.println(o1);
    }
}
```

Classes internes locales Commentaires sur l'exemple

- Noter la déclaration "final" du paramètre "v" et de la variable locale "tmp" dans la méthode setX() pour qu'elles puissent être utilisées dans la classe locale.
- Noter le type de retour de setX() : "ClasseLocale" n'est pas visible ; il faut donc mettre un super-type. Comme ici, notre classe n'implante pas d'interface et n'hérite pas d'une autre classe que "Object", on ne peut mettre autre chose que "Object".
- Au niveau de l'affichage, noter que les 2 affichages de "o1" donnent la même chose, et que celui de "o2" est différent. Cela montre que l'instance de la classe locale est créée avec une "clôture" des variables locales du bloc englobant.

Classes internes anonymes

- Déclaration
  - Après un new, dans une expression requérant une instance de classe
  - New SuperType(...) {...} où SuperType peut être une classe ou une interface.
  - Dans le cas où l'on souhaite passer par un constructeur du super-type autre que le constructeur sans argument, mettre les arguments requis entre les parenthèses
- Visibilité : hors de propos
- Utilisation
  - Lors de la création, là où la classe est déclarée
- Particularité
  - Sans utiliser la réflexivité, une seule instance possible
  - Implantation d'une interface au maximum.
  - Implantation de zéro interface si super-classe /= Object
- Intérêt
  - Alléger le code en évitant la multiplication de classes "visibles"

Classes internes anonymes Exemple

```
import java.awt.event.ActionEvent;import java.awt.event.ActionListener;
import java.util.ArrayList;

public class ClasseEnglobanteAnonyme {
    private int x; private ArrayList<ActionListener> al;
    public ClasseEnglobanteAnonyme() {al = new ArrayList<ActionListener>();}
    public void addActionListener(ActionListener l) {al.add(l);}
    public void fire() {for (ActionListener l : al) {l.actionPerformed(null);}}
    public void setX(final int v) {
        x = v;
        addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent ae) {System.out.println("x fixé à "+v);}
        });
    }
    public static void main(String[] args) {
        ClasseEnglobanteAnonyme cea = new ClasseEnglobanteAnonyme();
        cea.setX(2);        cea.setX(4);        cea.setX(6);        cea.fire();
    }
}
```

Classes internes anonymes Commentaires sur l'exemple

- Noter la déclaration du paramètre "v" de setX en "final"
- Noter la construction de l'instance et la déclaration :

  New ActionListener()

- Affichage produit :
  - x fixé à 2
  - x fixé à 4
  - x fixé à 6