IFT1025 été 2010

Miklós Csűrös

7 juin 2010

4 Classes imbriquées

En Java, on peut définir des **classes imbriquées** (*nested class*) : il s'agit d'une classe dont la déclaration se trouve dans le corps d'une autre classe (*outer class*). La classe externe a accès à tous les membres d'une classe imbriquée (même ceux déclarés privés). La classe imbriquée peut être statique : dans ce cas-là il s'agit d'une convénience de définir une hiérarchie. L'exemple ci-dessous définit la classe `java.awt.geom.Point2D.Float`.

```
package java.awt.geom;
public class Point2D
{
    protected Point2D(){...}; // pas d'instantiation directe
    ... // méthodes de la classe
    public static class Float extends Point2D
    {
       public Float(){...}; // instantiation OK
    }
}
```

⋆ règles d'accès dans la classe statique imbriquée : comme une méthode statique

4.1 Inner class

Une possibilité puissante est d'utiliser des classes imbriquées *non-statiques* : elles sont les **classes internes** (*inner class*). Une classe interne est le membre d'une *instance* de la classe imbriquante. Une classe interne peut être locale ou anonyme.

```
public class Outer
{
    private int x; // membre d'une instance
    public class Inner
    {
        private void increaseX() // accès à l'instance extérieure
        { x++; }
    }
    private void methode()
    {
        Inner inn = new Inner(); // associé avec cette instance (this)
    }
    public static void main(String[] args)
    {
        Outer ext = new Outer();
        Outer.Inner inn = ext.new Inner(); // instance Outer pour instantiation
    }
}
```

⋆ règles d'accès : comme une méthode dynamique
⋆ pas de membres statiques dans la classe interne

4.2 Local class

Une classe interne peut être définie dans le corps d'une méthode : il s'agit d'une **classe locale** (*local class*). Les règles de visibilité sont comme pour une variable locale : on peut utiliser la classe dans le corps de la méthode à partir de la fin de sa définition.

```
import javax.swing.JPanel;
import javax.swing.JButton;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class MonPanel extends JPanel
{
    public MonPanel(){super(); initComponents();}
    private void initComponents() // on place un bouton dans le milieu
    {
        JButton bouton = new JButton("Cancel");
        class ButtonListener implements ActionListener // classe locale
        {
            public void actionPerformed(ActionEvent ignored)
            {
                doCancel();
            }
        }
        bouton.addActionListener(new ButtonListener());
        this.add(bouton);
        ...
    }
    ... // methode doCancel() et d'autres membres
}
```

4.3 Anonymous class

Finalement, on peut aussi utiliser des **classes anonymes** (*anonymous class*). Une classe anonyme est utilisée lors de sa définition pour créer une seule instance. Syntaxe pour l'utilisation :

```
Outer objet = new Outer(...){ ... }
```

Entre les accolades, on peut définir les méthodes dont on a besoin. Le type `Outer` peut être une classe ou un interface.

- ⋆ si `Outer` est une **classe**, il est permis de redéfinir les méthodes héritées (*overriding*)
- ⋆ si `Outer` est un **interface**, il faut définir les méthodes de l'interface — c'est l'usage typique d'une classe anonyme

```
import javax.swing.JPanel;
import javax.swing.JButton;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class MonPanel extends JPanel
{
    public MonPanel(){super(); initComponents();}
    private void initComponents() // on place un bouton dans le milieu
    {
        JButton bouton = new JButton("Cancel");
        class ButtonListener implements ActionListener // classe locale
        {
        }
        bouton.addActionListener(new ActionListener()
            { // implémentation obligatoire de l'interface ActionListener
                public void actionPerformed(ActionEvent ignored)
                {
                    doCancel();
                }
            }
        );
        this.add(bouton);
        ...
    }
    ... // methode doCancel() et d'autres membres
}
```

- ⋆ pour accéder à l'instance extérieure dans une classe interne : écrire `Outer.this`.
- ⋆ pour accéder à une variable locale dans une classe interne : il faut déclarer la variable `final` (c-à-d juste une affectation permise)