Chapitre 4

Programmation orientée objet en Java

Comme nous l'avons déjà dit, le langage Java est un langage orienté presque pur. Il est seulement « presque » pur car il a quand même des types élémentaires (même si des classes correspondent également à ces types). Puisque ce langage est orienté objet (presque) pur, les unités logiques sont les *classes*.

4.1 Classes et objets en Java

Nous avons vu ce qu'est une *classe*, comment la mettre en place et comment l'utiliser à propos du langage C++. On conserve la même syntaxe en Java, à quelques variantes près.

4.1.1 Définition d'une classe

En Java, il n'y a que des définitions de classe, pas de déclaration. La définition d'une classe comprend l'**en-tête** et le **corps** de la classe.

4.1.1.1 En-tête d'une classe

Fonction.- L'en-tête de classe sert à donner un nom à la classe et à fixer quelques caractéristiques générales : sa visibilité, la classe mère (celle dont elle hérite) en cas d'héritage et une *interface* en cas d'héritage multiple.

Syntaxe.- La syntaxe de la définition est la suivante :

```
modificateur nom [extends superclass] [implements interfaces]
{
/* corps */
}
```

où le modificateur (ou attribut de visibilité) spécifie les conditions dans lesquelles la classe est

accessible, où l'indication éventuelle d'une superclasse indique la classe dont elle hérite et où l'indication éventuelle d'interfaces correspond à la mise en place de l'héritage multiple en Java.

Les modificateurs de classe.- Les modificateurs possibles sont :

- **public** pour qu'elle soit accessible par tous ;

- **private** pour qu'elle ne soit accessible que dans le fichier où elle est définie ;

- **final** pour qu'elle ne puisse plus être modifiée, c'est-à-dire redéfinie ; aucune classe ne peut alors hériter de cette classe ;

- **abstract** si elle est définie dans un autre langage, tel que C++ (il faut bien quelques classes pour commencer) ;

- **synchronizeable** que nous verrons lorsque nous aborderons le multitâche.

4.1.1.2 Corps d'une classe

Le corps d'une classe est composé de **membres**. Les membres sont de deux sortes :

- les **champs** (qui correspondent aux membres données du C++) sont des variables ;

- les **méthodes** (qui correspondent aux fonctions membres du C++) sont constituées d'instructions.

4.1.1.3 Constructeur

Définition 1.- *Un* **constructeur** *d'une classe est une méthode dont le nom est le nom de cette classe et qui ne possède pas de type de retour.*

Définition 2.- *Le* **constructeur par défaut** *d'une classe est le constructeur sans argument.*

Remarque.- Un constructeur se définit donc comme en C++. Par contre, en Java, il existe toujours un constructeur par défaut. Si le concepteur de la classe ne le définit pas explicitement, le compilateur Java lui en attribue un par défaut.

Remarque sur le destructeur.- Contrairement au langage C++, il n'y a pas de destructeur en Java. Le **ramasse-miettes** (*garbage collector* en anglais) se charge de récupérer la place dans la mémoire centrale.

4.1.2 Un exemple de classe

Définissons une classe `point` :

```
// point.java

public class point
   {
   private float x, y;

   public point()
      {
      x = y = 0;
      }
```

```
   public void initialise(float abs, float ord)
      {
      x = abs;
      y = ord;
      }
   public void deplace(float dx, float dy)
      {
      x = x + dx;
      y = y + dy;
      }
   public String affiche()
      {
      String s;
      s = new String();
      s = "Le point est en (" + x + "," + y + ")";
      return s;
      }
   }
```

4.1.3 Les objets en Java

4.1.3.1 Instantiation de classe

Les **objets** sont les éléments des classes.

Déclaration.- La déclaration d'un objet se fait comme pour n'importe quelle variable, la différence étant que le type est une classe :

```
type nom ;
```

Instantiation.- Tout objet doit être instantié avant de pouvoir être utilisé, c'est-à-dire qu'on doit lui appliquer un constructeur. Ceci se fait de la façon suivante :

```
nom = new type(arguments) ;
```

4.1.3.2 Un exemple

Utilisons notre classe `point` dans une applet :

```
// testpoint.java

import java.applet.Applet;
import java.awt.Graphics;

public class testpoint extends Applet
   {
   private point A;

   public void paint(Graphics g)
      {
      A = new point();
      g.drawString(A.affiche(), 25, 25);
```

```
    A.initialise(2.34f, 5.1f);
    g.drawString(A.affiche(), 25, 40);
    A.deplace(1.03f, 3.3f);
    g.drawString(A.affiche(), 25, 55);
    };
}
```

Remarque.- Il est inutile d'importer la classe `point` si elle se trouve dans le même répertoire.

4.2 Héritage simple

4.2.1 La superclasse `Object`

Toutes les classes de Java dérivent de la classe `Object`.

4.3 Interfaces

4.4 Surcharge

4.5 Polymorphisme

4.6 Classes abstraites