Algorithmique Avancée
Outils JAVA : Classes abstraites, Interfaces

Serigne A. Gueye

Septembre 2014, CERI

1 Introduction

L'objet de ce document est de pointer succintement un certain nombre de concepts objets et d'outils JAVA intervenant précisement dans les travaux pratiques à réaliser. La lecture d'un des nombreux ouvrages JAVA est nécessaire pour avoir une vue exhaustive. Vous pouvez par exemple vous référez au livre très complet "Programmer en JAVA", Claude Delannoy, Editions Eyrolles.

2 Classes abstraites

Définition 2.1 *Une classe* **abstraite** *est une classe que l'on ne peut pas instancier. Elle sert de base à une dérivation (ou héritage).*

Exemple .

```
abstract   class   A
{
           .       .
}
```

□

On peut trouver dans une classe abstraite des champs et des méthodes dont héritera toute classe dérivée de celle-ci.

Définition 2.2 *Une méthode* **abstraite** *est une méthode dont on ne fournit que la signature lors de la déclaration de la classe.*

Exemple .

```
abstract   class     A
{
           public    abstract void affiche() ;
}
```

affiche() est ici une méthode abstraite de la classe A.

A ne peut pas être instanciée : i.e A obj = new A() ne marchera pas.

```
abstract   class     B extends A
{
           int       info ;
           public    B()
           {
                     info = 0 ;
           }

           public    void affiche()
           {
                     System.out.println("info = " + info) ;
           }
}
```

La classe B hérite de A et implémente la méthode abstraite affiche().

B obj = new B() est possible.

□

Propriété 2.3 *Une classe dérivée d'une classe abstraite peut être instanciée si et seulement si toutes les méthodes abstraites ont été définies.*

Notez que dès lors qu'une méthode a été déclarée abstraite la classe la contenant le devient aussi, même si le mot clé "abstract" n'est pas ajouté à la classe.

3 Interfaces

Définition 3.1 *Une* **interface** *est une variante de classe abstraite. Elle ne comporte que des méthodes qui sont toutes abstraites, ou des constantes.*

L'interface va permettre de mettre en oeuvre **l'héritage multiple** dans lequel une classe pourra dériver de plusieurs classes de base. L'héritage multiple n'est pas possible avec les classes abstraites.

Exemple . La définition d'une interface est similaire à celle d'une classe.

```
interface   class     I
{
            public    void affiche() ;
}
```

On définit ici une interface I comportant une seule méthode abstraite affiche().

```
class   C         implements I
{
        int       x ;

        public    void affiche()
        {
                  System.out.println("x = " + x) ;
        }
}
```

La classe C implémente l'interface I en redéfinissant la méthode affiche().

□

La clause "implements" est une garantie assurant qu'une classe implémentera les méthodes d'une interface. Notons que toutes les méthodes d'une interface doivent être implémentées.

La clause est indépendante de l'héritage, c'est à dire du mot clé "extends". La classe C aurait donc pû tout à la fois dériver de A et implémenter I.

4 Classes génériques à un seul type

Définition 4.1 *Une classe* **générique** *à un seul paramètre T (de type classe), est une classe permettant d'agir sur n'importe quel objet de type T.*

La classe est définie en indiquant entre crochets le paramètre T comme suit :

```
class   G<T>
{
        T       x;

        public  G(T x)
        {
                this.x = x;
        }
        public  void affiche()
        {
                System.out.println("x = " + x);
        }
}
```

Le nom du paramètre est T.

Il est important de souligner que cette déclaration est en fait équivalente à :

```
class   G
{
        Object  x;

        public  void affiche()
        {
                System.out.println("x = " + x);
        }
}
```

où “Object“ est la super classe de base JAVA.

Si nous déclarons, Integer o = 3, puis que nous écrivons, G<Integer> g = new G<Integer>(o), le compilateur le traduira en efffectuant une conversion du type “Object” en “Integer”, car par les régles de dérivation un “Object” est aussi un “Integer”.

Le type T étant une classe peut faire l'objet de mécanisme d'héritage et d'implémentation d'interface du même type que ceux présentés ci-dessus.

De la même façon que pour les classes, il est possible de définir des **interfaces génériques**. L'une d'elle est particulièrement utile par rapport à nos TP : l'interface prédéfinie **“Comparable<T>”**.

Elle est utile quand on est amené à créer des objets que l'on devra comparer entre eux. Elle dispose d'une méthode abstraite **“compareTo(T o)”** dont l'implémentation permet de décrire comment deux objets doivent être comparés.