Chapitre 13

Compléments en vrac : facilités d'écriture en java

1 Les types génériques

Les classes génériques ont été introduites à partir de la version 1.5 de java.

Les méthodes en java sont *paramétrées*. Les paramètres sont fournis entre les parenthèses qui suivent le nom de la méthode. Lorsque la méthode est invoquée, le calcul qu'elle réalise se fait sur les valeurs qu'elle a reçues en paramètres.

Les types génériques constituent une analogie à ce mécanisme, mais pour les classes. Les « paramètres » que l'on fournit à une classe ne sont pas des valeurs, mais des *types*. Le type des éléments que va manipuler une classe générique est précisé au moment de l'instanciation de la classe.

1.1 Utilisation de types génériques existants

Un certain nombre de classes définies dans l'API java sont génériques. C'est typiquement le cas pour une collection, qu'on va pouvoir paramétrer par le type des éléments qu'elle va rassembler.

1.1.1 Déclaration

Le type est fourni en paramètre en le plaçant entre les symboles '<' et '>' juste derrière le nom de la classe.

Par exemple, pour déclarer une ArrayList nommée listeHoraires de Horaire :

```
ArrayList<Horaire> listeHoraires;
```

On ne pourra pas mettre n'importe quel Object dans une telle liste, seulement des Horaire ou un type dérivé, comme HorairePrecis.

Autre exemple, pour déclarer une ArrayList nommée listeFormes de FormePositionnable :
ArrayList<FormePositionnable> listeFormes ;

1.1.2 Instanciation

Pour l'instanciation, on précise également le type entre les symboles '<' et '>' juste après le nom de la classe générique, et avant l'ouverture des parenthèses.

Par exemple, pour instancier listeHoraires :
listeHoraires = new ArrayList<Horaire>() ;
ou pour listeFormes :
listeFormes = new ArrayList<FormePositionnable>() ;

N.B. On peut bien sûr toujours réaliser la déclaration et l'instanciation dans la même ligne :
ArrayList<Horaire> listeHoraires = new ArrayList<Horaire>() ;
ArrayList<FormePositionnable> listeFormes = new ArrayList<FormePositionnable>() ;

1.1.3 Interfaces génériques

Les interfaces sont des types en java, et peuvent donc aussi être génériques. Ainsi, l'interface Comparable est générique.

Revenons sur l'exemple considéré dans la partie 3.1 page 70 de ce cours. Une classe A implantait l'interface Comparable et on avait dû écrire la méthode compareTo(Object o) avec beaucoup de transtypage.

Avec les types génériques, on aurait pu écrire A de manière beaucoup plus élégante par :

```
public class A implements Comparable<A> {
    public int x, y ;

    public int compareTo(A that) {
      return (this.x + this.y) - (that.x + that.y) ;
    }
}
```

D'une manière générale, les génériques évitent beaucoup de transtypage.

1.2 Écrire une classe générique

On peut soit même écrire une classe générique. Il suffit de faire suivre dans sa définition le nom de la classe par '<T>', où T est le nom que l'on choisit pour désigner le type qui sera

reçu en paramètre. Dans le corps de la classe, on peut alors utiliser la notation T chaque fois qu'on a besoin de désigner ce type dans le code. Si on veut paramétrer par plusieurs types, on les sépare par des virgules entre les symboles '<' et '>', comme par exemple dans <T, E>.

Exemple. Une classe générique FIFO qui implante une file d'attente (FIFO est l'acronyme pour *First In First Out*).

```
import java.util.ArrayList;

public class FIFO<T> {
    private ArrayList<T> fifo;

    public FIFO() {
      fifo = new ArrayList<T>();
    }

    public void push(T e) {
      fifo.add(e);
    }

    public T pop() {
      if (! this.isEmpty())
          return fifo.remove(0);
      else
          return null;
    }

    public T see() {
      // pour voir l'element en tete de file sans le sortir de la file
      if (! this.isEmpty())
          return fifo.get(0);
      else
          return null;
    }

    public int size() {
      return fifo.size();
    }

    public boolean isEmpty() {
      return fifo.size() == 0;
    }
}
```

2 Méthodes à nombre de paramètres variables : utilisation de *l'ellipse*

Cette notation a également été introduite par java 1.5.

L'*ellipse* est le nom en français de la notation trois petits points : « ... ». Elle est autorisée en java pour indiquer qu'une méthode accepte un nombre variable de paramètres. Par exemple, la notation

```
int... valeurs
```

sera utilisée pour désigner qu'un nombre variable de paramètres de type int est admis par une méthode.

Par exemple, la méthode toto suivante accepte un paramètre de type String, puis un nombre variable de paramètres de type int :

```
void toto(String s, int... valeurs)
```

Les invocations toto("Bonjour", 1, 2, 3) ou toto("Salut", 1, 8, 16, 76, 23, -2, 0), ou encore toto("Hello") (zéro paramètre de type int fourni) sont valides.

A l'intérieur de la méthode toto, on manipule valeurs comme si c'était un tableau, avec la syntaxe à doubles crochets. Par exemple, pour afficher dans toto toutes les valeurs entières passées en paramètre par le biais du paramètre valeurs :

```
void toto(String s, int... valeurs) {
    for (int i=0; i<valeurs.length; i++)
        System.out.println(valeurs[i]);
}
```

Les restrictions sont que :

- une seule ellipse est autorisée par méthode ;
- l'ellipse doit obligatoirement être en dernière position dans la liste des paramètres.

Exemple. Une méthode (statique) qui calcule la moyenne d'un nombre variable de paramètres entiers :

```
static double moyenne(int... valeurs) {
    int somme = 0;
    for (int i=0; i < valeurs.length; i++)
        somme = somme + valeurs[i];
    return somme/valeurs.length;
}
```

3 Les boucles *for each*

Pour itérer sur tous les éléments d'une collection ou d'un tableau, java (depuis java 1.5) propose une syntaxe particulière qu'on appelle la boucle *for each*, et qui permet de rendre implicite l'utilisation d'un itérateur (ou des indices dans le cas d'un tableau).

La syntaxe est
for (x : $nomCollec$)
où $nomCollec$ est le nom de la collection à parcourir, et x est le nom de la variable qui prendra successivement toutes les valeurs contenues dans $nomCollec$. La variable x doit être typée par le type des éléments contenus dans $nomCollec$.

Quelques exemples d'utilisation

Exemple. Soit listeHoraires une ArrayList de Horaire (listeHoraires est donc de type ArrayList<Horaire>).

Pour afficher tous les horaires contenus dans listeHoraires :

```
for (Horaire h : listeHoraires)
    System.out.println(h);
```

Exemple. Pour faire la somme de toutes les valeurs contenues dans un tableau de int nommé tab (tab est de type int[]) :

```
int somme = 0;
for (int n : tab)
    somme = somme+n;
```

Attention. Dans l'écriture for (int n : tab), la variable n ne doit pas être confondue avec un indice i qui parcourerait le tableau. n est de type int car les valeurs du tableau sont de type int.

Exemple. Utilisation de la boucle *for each* et de l'ellipse dans une méthode statique qui calcule la moyenne d'une série de taille variable de nombre réels.

```
static double moyenne(double... valeurs) {
    double somme = 0;
    for (double v : valeurs)
        somme = somme + v;
    return somme/valeurs.length;
}
```