Chapitre 13

Les paquetages

1. Présentation

Les paquetages («*packages*») servent à structurer l'ensemble des classes écrits en Java. Leur utilité est multiple :

- Faciliter la recherche de l'emplacement physique des classes quand elles sont nécessaires (pour la compilation d'un fichier ou pour l'exécution d'un programme).
- Empêcher la confusion entre des classes de même nom.
- Structurer l'ensemble des classes selon une arborescence, ce qui rend beaucoup plus lisible l'ensemble.

Java utilise fortement le principe des packages pour regrouper les classes de base du langage :

java.applet
Classes de base pour les applets

java.awt
Classes d'interface graphique AWT

java.io
Classes d'entrées/sorties (flux, fichiers)

java.lang ← **importée automatiquement par le compilateur**
Classes de support du langage

java.math
Classes permettant la gestion de grands nombres.

java.net
Classes de support réseau (URL, sockets)

java.rmi
Classes pour les méthodes invoquées à partir de machines virtuelles non locales.

java.security
Classes et interfaces pour la gestion de la sécurité.

java.sql
Classes pour l'utilisation de JDBC.

java.text
Classes pour la manipulation de textes, de dates et de nombres dans plusieurs langages.

java.util
Classes d'utilitaires (Scanner, Vector, Hashtable, ...)

javax.swing
Classes d'interface graphique

2. Nommage

Forme de nom de paquetage :

`coursJava.monPaquetage`

Si la classe **`Etudiant`** se trouve dans un paquetage **`coursJava.monPaquetage`**, alors son nom complet est **`coursJava.monPaquetage.Etudiant`**.

Le fichier **`Etudiant.class`** devra nécessairement être rangé dans un répertoire de nom **`monPaquetage`** contenu dans un répertoire de nom **`coursJava`**.

→ La fin du chemin d'accès à ce fichier sera **`coursJava/monPaquetage`**.

3. Instruction `import`

Pour utiliser une classe issue d'un autre paquetage, on devrait normalement spécifier son nom complet pour indiquer son emplacement :

```
public class Classe
{
   private java.util.Vector liste;

   public Classe()
   {
      liste = new java.util.Vector();
   }
}
```

Pour éviter de nommer complètement les classes issues d'un autre paquetage, on utilise l'instruction **`import`** :

```
import java.util.Vector;

public class Classe
{
   private Vector liste;

   public Classe()
   {
      liste = new Vector();
   }
}
```

Remarque

- En écrivant :

```
import java.util.Vector;
```

on n'importe que la classe **`Vector`** définie dans le paquetage **`java.util`**.

- En écrivant :

```
import java.util.*;
```

on importe toutes les classes définies dans le paquetage **`java.util`**.

Remarque

```
import java.util.*;
```

L'instruction **import** ne va pas importer de manière récursive les classes se trouvant dans **java.util** et dans ses sous paquetages. Elle va uniquement se contenter de faire un balayage d'UN SEUL niveau. Elle ne va donc importer que les classes du paquetage **java.util**.

```
import java.awt.*;
import java.awt.event.*;
```

La première instruction importe toutes les classes se trouvant dans le paquetage **awt**. Elle ne va pas importer par exemple les classes se trouvant dans le sous paquetage **event**. Si vous avez besoin d'utiliser les classes de **event**, vous devez les importer aussi.

Remarque

Si on utilise dans un même fichier Java les classes **`coursJava.monPaquet.Etudiant`** et **`coursJava.tonPaquet.Etudiant`**, alors les directives :

```
import coursJava.monPaquet.Etudiant;
```

et

```
import coursJava.tonPaquet.Etudiant;
```

ne peuvent pas servir à faire un raccourci en nommant l'une ou l'autre des classes par **`Etudiant`** : on est obligé d'utiliser les noms complets.

Remarque

A l'intérieur d'un même paquetage, on peut utiliser directement toute classe de ce paquetage sans instruction import.

4. Création d'un paquetage

Considérons la classe suivante, qui se trouve dans le fichier Etudiant.java.

```
package coursJava.monPaquet;

public class Etudiant
{
   private String nom;
   public void afficher()
   {
      System.out.println(nom);
   }
}
```

```
package coursJava.monPaquet;

public class Etudiant
{
   private String nom;
   public void afficher()
   {
      System.out.println(nom);
   }
}
```

La première ligne indique que cette classe fait partie d'un paquetage.

L'instruction **`package`**, si elle existe, doit toujours être en première ligne du code.

Le fichier **`Etudiant.java`** peut se trouver n'importe où, mais il est nécessaire que la fin du chemin d'accès au fichier **`Etudiant.class`** soit **`coursJava/monPaquet/`**

Il faut donc veiller à mettre les fichiers **`.class`** dans le bon répertoire :

- soit mettre le fichier **`.java`** dans le même répertoire avant de le compiler ;
- soit utiliser l'option -d de la commande javac pour ranger le fichier **`.class`** à sa place au moment de la compilation ;
- soit effectuer un déplacement "manuel".

5. Utilisation d'un paquetage

Considérons la classe suivante qui se trouve dans un répertoire quelconque :

```
import coursJava.monPaquet.Etudiant;

class UtiliseClasse
{
   public static void main(String[] arg)
   {
      Etudiant etud = new Etudiant();
      etud.afficher();
   }
}
```

Lorsqu'on veut compiler cette classe, le problème pour le compilateur est de trouver la classe **`coursJava.monPaquet.Etudiant`**.

Il faut lui indiquer le chemin. Une partie de cette indication est donnée par l'instruction **`import`**, qui indique sur notre exemple que la classe **`Etudiant`** se trouve dans le répertoire **`monPaquet`** et que ce répertoire se trouve dans le répertoire **`coursJava`**.

→ Mais il faut que le compilateur connaisse le reste du chemin.

1) Option **`-classpath`** *de la commande* **`javac`**

On indique à la suite de **`-classpath`** les chemins des répertoires contenant les classes nécessaires à la compilation (en les séparant par des ':', ou des ';').

Ici, supposons que le répertoire **`coursJava`** soit dans le répertoire **`c:\mesPaquetages`**. La commande sera :

```
javac -classpath c:\mesPaquetages UtiliseClasse.java
```

ou :

```
javac -cp c:\mesPaquetages UtiliseClasse.java
```

```
javac -cp c:\mesPaquetages;c:\autreRep\ UtiliseClasse.java
```

Le compilateur cherchera alors à partir du répertoire **`c:\mesPaquetages`** dans **`coursJava\monPaquet`**, comme indiqué par le nom du paquetage, et trouvera la classe **`Etudiant`**.

2) Variable d'environnement **`CLASSPATH`**

On peut placer dans cette variable le chemin d'accès au répertoire racine du paquetage, par exemple le répertoire **`c:/coursJava`**.

Ainsi, lorsque le compilateur constate qu'il doit importer la classe **`coursJava.monPaquet.Etudiant`**, il cherche à partir du chemin suggéré par la variable **`CLASSPATH`** le fichier de chemin relatif :

`c:/coursJava/monPaquet/Etudiant.class`

6. L'utilitaire JAR (Java ARchive)

Il est possible de créer une enveloppe qui va contenir tous les fichiers d'une application Java ou une portion de cette application dans un fichier d'extension **.jar**.

Ceci inclus : l'arborescence des packages, les fichiers .class, les fichiers de ressources (images, configuration, ...), ...

Un fichier .jar est physiquement une archive de type Zip qui contient tous ces éléments.

Création d'un JAR

L'utilitaire **jar.exe** utilisable en ligne de commande permet de créer une archive JAR ou de lire le contenu d'une archive existante :

```
jar {ctx}[vf0] [jar-file] files ...
```

Options:

- c crée une nouvelle archive
- t liste le contenu d'une archive
- x restitue les fichiers nommés (tous par défaut) d'une archive
- v génère un affichage sur la sortie standard d'erreur
- f spécifie le nom de l'archive
- 0 annule la compression de l'archive

Exemple : pour archiver deux fichiers (f1.class et f2.class) dans une nouvelle archive (a1.jar) :

```
jar cvf a1.jar f1.class f2.class
```

Utilisation d'un JAR lors de la compilation

On peut utiliser des fichiers .jar sur la ligne de compilation de javac :

```
javac -classpath c:\mesPaquetages\lib.jar prog.java
```

Remarque : avec **-classpath** on donne le nom du fichier .jar, alors que dans le cas de packages on indique le nom du répertoire qui contient les fichiers .class

Exécution d'un JAR

Le fichier .jar peut être exécuté s'il contient au moins une classe avec une méthode main() :

```
java  -cp  MonApplication.jar  ClassePrincipale
```

Le fichier jar peut inclure un fichier **manifest** qui permet de préciser des informations d'exécution sur le fichier jar (classe principale à exécuter, classpath, …).

manifest.mf

```
Main-Class: coursJava.tests.ClassePrincipale
Class-Path: repertoire/de/class
```

Lors de la création du fichier .jar il faut inclure le fichier manifest.mf dans le jar :

jar cfm MonApplication.jar f1.class f2.class manifest.mf

Ceci permet d'exécuter directement l'application en double-cliquant sur le fichier .jar ou en tapant l'instruction :

java -jar MonApplication.jar