Chapitre 7

Les packages en Java

Définition : Java permet de regrouper les classes en ensembles appelés packages afin de faciliter la modularité.

Parmi les paquetages de Java vous avez :

`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`	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 (vecteurs, hashtable)
`javax.swing`	Classes d interface graphique

Caractéristiques

Nom: chaque paquetage porte un nom. Ce nom est soit un simple identificateur ou une suite d'identificateurs séparés par des points. L'instruction permettant de nommer un paquetage doit figurer au début du fichier source (.java) comme suit:

```
------------- Fichier Exemple.java ----------
package nomtest; // 1ere instruction

class MTest {}

class Uneautre {}

public class Exemple {// blabla suite du code ...}

------------- Fin du Fichier ---------------
```

Ainsi par cette opération nous venons d'assigner toutes les classes du fichier Exemple.java au paquetage nomtest.

Si l'instruction package était absente du fichier, alors c'est le paquetage par défaut qui est pris en considération. Ainsi toutes les classes du le dit fichier vont appartenir au paquetage par défaut.

Utilisation: Si vous instanciez un objet d'une classe donnée, le compilateur va chercher cette classe dans le fichier où elle a été appelée, nous dirons que cette recherche a eu lieu dans le paquetage par défaut même si ce dernier porte un nom.

Si votre appel fait référence à une classe appartenant à un autre paquetage, vous devez aider le compilateur à retrouver le chemin d'accès vers cette classe en procédant comme suit:

- En mentionnant le chemin complet d'accès à la classe : En citant les noms complets des paquetages nécessaires:

```
nomtest.Uneautre mm = new nomtest.Uneautre();
```

C'est une opération fastidieuse si vous avez affaire à une arborescence de paquetages ou bien à un nombre important de classes dans un même paquetage. Source d'erreurs donc.

- En important les paquetages utiles : Vous devez utiliser pour cela l'instruction import comme suit:

```
------------- Fichier Test.java ------------
import nomtest.Uneautre;


public class Test {
       // balblabla ...
       Uneautre mm = new Uneautre();
}

// blablabla ....
----------------Fin Fichier-----------------
```

C'est bien beau! Mais si vous avez besoin d'utiliser aussi une autre classe se trouvant dans le paquetage nomtest. Vous pouvez le faire soit en important nomtest comme dans le cas de la classe Uneautre ou bien écrire ce qui suit:

```
import nomtest.*;
```

Par cette instruction vous avez demandé d'importer toutes les classes se trouvant dans le paquetage nomtest.

Remarques

-1- Le paquetage java.lang est importé automatiquement par le compilateur.

-2- import nomtest.*;
Cette instruction ne va pas importer de manière récursive les classes se trouvant dans nomtest et dans ses sous paquetages. Elle va uniquement se contenter de faire un balayage d'un **SEUL** niveau. Elle va importer donc que les classes du paquetage nomtest.

```
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.

-3- Si deux paquetages contiennent le même nom de classe il y a problème! Par exemple la classe List des paquetages java.awt et java.util ou bien la classe Date des paquetages java.util et java.sql etc.

Pour corriger ce problème vous devez impérativement spécifier le nom exact du paquetage à importer (ou à utiliser).

```
java.awt.List (ou) java.util.List ; java.util.Date (ou) java.sql.Date

import java.awt.*; // ici on importe toutes les classes dans awt
import java. util.*; // ici aussi
import java.util.List; // ici on précise la classe List à utiliser
```

-4- Vous pouvez regrouper toute une hiérarchie de classes (ou un seul répertoire) dans un fichier ZIP ou JAR (grosso modo contient les mêmes options que la commande unix: tar)

```
zip montest.zip montest/*
jar -cvf montest.jar montest/*
```

c: pour créer ; v: pour un affichage détaillé (verbose) ; f: pour préciser le nom de fichier

Par la suite vous pouvez préserver dans un endroit unique par exemple le répertoire lib tous les zip ou jar. Si vous procédez ainsi, vous devez indiquer au compilateur comment accéder au répertoire lib en précisant son chemin exact comme suit:

- À travers la ligne de commande en utilisant l'option -classpath

```
javac -classpath chemin1 nom_fic.java
```

- En définissant la variable d'environnement CLASSPATH:

Dans le fichier .cshrc:

```
setenv CLASSPATH .
setenv CLASSPATH ${CLASSPATH}:UnChemin/lib/montest.zip
```

Dans le fichier Autoexec.bat (dans Windows, sinon définir une variable système si vous utilisez xp ou 2000). Attention dans Windows les ":" sont remplacées par ";".

```
set CLASSPATH=.
set CLASSPATH %CLASSPATH%;UnChemin/lib/montest.zip
```

Exemple Pratique

Étudier les paquetages fournis dans le fichier TestPackage.zip