PAQUETAGES

Paquetages: Définitions

- Les classes Java sont regroupées en paquetages (*packages* en anglais)
- Ils correspondent aux « bibliothèques » des autres langages comme le langage C, Basic,Fortran, etc...
- Les paquetages offrent un niveau de modularité supplémentaire pour
  - réunir des classes suivant un centre d'intérêt commun
  - la protection des attributs et des méthodes
- Le langage Java est fourni avec un grand nombre de paquetages

Quelques paquetages du SDK

- **java.lang** : classes de base de Java
- **java.util** : utilitaires
- **java.io** : entrées-sorties
- **java.awt** : interface graphique
- **javax.swing** : interface graphique avancée
- **java.applet** : applets
- **java.net** : réseau
- **java.rmi** : distribution des objets

Nommer une classe

- Le nom complet d'une classe (*qualified name*

dans la spécification du langage Java) est le nom

de la classe préfixé par le nom du paquetage :

java.util.ArrayList

- Une classe du même paquetage peut être

désignée par son nom « terminal » (les classes du

paquetage **java.util** peuvent désigner la classe ci-dessus par « **ArrayList** »)

- Une classe d'un autre paquetage doit être

désignée par son nom complet

Importer une classe d'un paquetage

- Pour pouvoir désigner une classe d'un autre paquetage par son nom terminal, il faut l'importer

```
import java.util.ArrayList;
public class Classe {
…
ArrayList liste = new ArrayList();
```

- On peut utiliser une classe sans l'importer ; l'importation permet seulement de raccourcir le nom d'une classe dans le code:

```
java.util.ArrayList Liste = new java.util.ArrayList();
```

Importer toutes les classes d'un paquetage

- On peut importer toutes les classes d'un paquetage :

  import java.util.*;

- Les classes du paquetage **java.lang** sont implicitement importées

Lever une ambiguïté

- On aura une erreur à la compilation si

– 2 paquetages ont une classe qui a le même nom

– ces 2 paquetages sont importés en entier

- Exemple (2 classes **List**) :

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

- Pour lever l'ambiguïté, on devra donner le nom complet de la classe. Par exemple,

```
java.util.List l = getListe();
```

Importer des constantes **static**

- Depuis le JDK 5.0 on peut importer des variables ou méthodes statiques d'une classe ou d'une interface
- On allège ainsi le code, par exemple pour l'utilisation des fonctions mathématiques de la classe **java.lang.Math**
- A utiliser avec précaution pour ne pas nuire à la lisibilité du code (il peut être plus difficile de savoir d'où vient une constante ou méthode)

Exemple d'**import static**

- Import static java.lang.Math.*;
- Import java.util.*;
- Public class Machin {

  …

  X = max(sqrt(abs(y)), sin(y)); // au lieu de Math.sqrt, Math.sin…
- On peut importer une seule variable ou méthode :
  - **Import static java.lang.Math.PI;**
  - X = 2* PI;

Ajout d'une classe dans un paquetage

- **package** *nom-paquetage*;

doit être la première instruction du fichier source définissant la classe (avant même les instructions **import**)

- Par défaut, une classe appartient au paquetage par défaut qui n'a pas de nom, et auquel appartiennent toutes les classes situées dans le même répertoire (et qui ne sont pas d'un paquetage particulier)

Sous-paquetage

- Un paquetage peut avoir des sous-paquetages
- Par exemple, **java.awt.event** est un sous-paquetage de **java.awt**
- L'importation des classes d'un paquetage

n'importe pas les classes des sous-paquetages ;

on devra écrire par exemple :

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

Nom d'un paquetage

- Le nom d'un paquetage est hiérarchique **:java.awt.event**
- Il est conseillé de préfixer ses propres paquetages par son adresse Internet :
  - **Com.oreilly.projets.LivresJava**

Placement d'un paquetage

- Les fichiers **.class** doivent se situer dans l'arborescence d'un des répertoires du *classpath*
- Le nom relatif du répertoire par rapport au répertoire du *classpath* doit correspondre au nom du paquetage
- Par exemple, les classes du paquetage **ma.ensat.toto.liste** doivent se trouver dans un répertoire **ma/ensat/toto/liste** relativement à un des répertoires du *classpath*

Encapsulation d'une classe dans un paquetage

- Si la définition de la classe commence par **public class**
  - la classe est accessible de partout
- Sinon, la classe n'est accessible que depuis les classes du même paquetage

Compiler les classes d'un paquetage

- **javac -d** *répertoire-package Classe.java*
- L'option « **-d** » permet d'indiquer le répertoire *racine* où sera rangé le fichier compilé
- Si on compile avec l'option « **-d** » un fichier qui comporte l'instruction « **package nom1.nom2**», le fichier **.class** est rangé dans le répertoire *répertoire-package*/**nom1**/**nom2**

Compiler les classes d'un paquetage

- Remarque:
  - Si on compile sans l'option « **-d** », le fichier **.class** est rangé dans le même répertoire que le fichier **.java** (quel que soit le paquetage auquel appartient la classe)

Compiler les classes d'un paquetage

- Quand on compile les classes d'un paquetage avec l'option **-d**, on doit le plus souvent indiquer où sont les classes déjà compilées du paquetage avec l'option **-classpath** :
- **javac -classpath** *répertoire-package*

-d *répertoire-package* **Classe.java**

Exécuter une classe d'un paquetage

- On lance l'exécution de la méthode **main()**

d'une classe en donnant son nom complet

(préfixé par le nom de son paquetage)

- Par exemple, si la classe **C** appartient au

paquetage **p1.p2** :

java p1.p2.C

- Le fichier **C.class** devra se situer dans un sous répertoire **p1/p2** d'un des répertoires du *classpath* (option **-classpath** ou variable **CLASSPATH**)

Utilisation pratique des paquetages

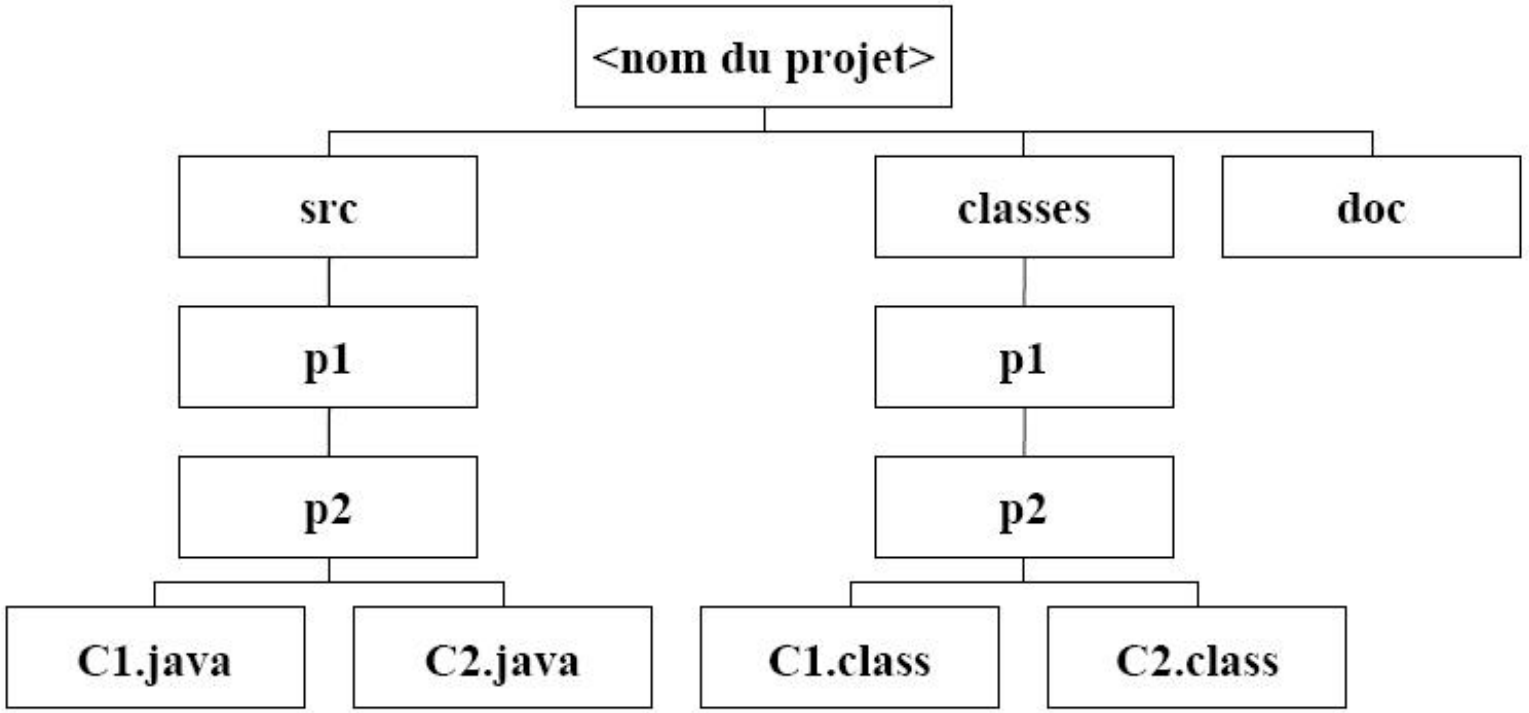
- Les premières tentatives de développement avec paquetages conduisent à de grosses difficultés pratiques pour compiler les classes
- Ces difficultés peuvent être évitées

– en respectant quelques principes simples pour le placement des fichiers sources et classes (exemple dans les transparents suivants)

– en utilisant correctement les options **–classpath** et **-d**

Placements préconisés pour le développement d'une application

- Classes du paquetage p1.p2



Commandes à lancer

- Si on se place dans le répertoire racine,

– pour compiler (*en une seule ligne*) :

javac –d classes –classpath classes
src/p1/p2/*.java

– pour exécuter :

java –classpath classes p1.p2.C1

– pour générer la documentation :

javadoc –d doc –sourcepath src p1.p2

- On peut ajouter d'autres répertoires ou fichier .jar dans le *classpath*

Classes inter-dépendantes

- Si 2 classes sont inter-dépendantes, il faut les indiquer toutes les deux dans la ligne de commande java de compilation :
- **javac ... A.java B.java**

ou

- **javac ... *.java**

Option **-sourcepath**

- javac peut ne pas savoir où sont les fichiers source de certaines classes dont il a besoin (en particulier si on compile plusieurs paquetages en même temps)
- L'option **–sourcepath** indique des emplacements pour des fichiers sources
- Comme avec *classpath*, cette option peut être suivie de fichiers jar, zip ou des répertoires racines pour les fichiers source (l'endroit exact où se trouvent les fichier **.java** doit refléter le nom du paquetage)

Utilisation des fichiers source

- javac recherche les classes dont il a besoin dans le *classpath*
- S'il ne trouve pas le fichier **.class**, mais s'il

trouve le fichier **.java** correspondant, il le compilera pour obtenir le **.class** cherché

- S'il trouve les 2 (**.class** et **.java**), il

recompilera la classe si le **.java** est plus récent

que le **.class**

Exemple avec **sourcepath**

- Situation : compiler une classe **C**, et toutes les classes dont cette classe dépend (certaines dans des paquetages pas encore compilés)
- « **javac C.java** » ne retrouvera pas les fichiers class des paquetages qui ne sont pas déjà compilés
- On doit indiquer où se trouvent les fichiers source de ces classes par l'option **–sourcepath** ; par exemple,

javac –sourcepath src –classpath classes
–d classes C.java

Problème avec **sourcepath**

- Si **C.java** a besoin d'un paquetage pas encore compilé et dont les sources sont indiquées par l'option **sourcepath**
- Si ce paquetage comporte une erreur qui empêche sa compilation
- Alors javac envoie un message d'erreur indiquant que le paquetage n'existe pas, sans afficher de message d'erreur de compilation pour le paquetage
- L'utilisateur peut alors penser à tort que l'option **sourcepath** ne marche pas

Classpath

- Le *classpath* contient par défaut le seul répertoire courant (il est égal à « . »)

- Si on donne une valeur au *classpath*, on doit indiquer le répertoire courant si on veut qu'il y soit

- Dans le *classpath* on indique des endroits où trouver les classes et interfaces :
  – des répertoires (les classes sont recherchées sous ces répertoires selon les noms des paquetages)
  – des fichiers **.jar** ou **.zip**

Exemples de Classpath

- Sous Unix, le séparateur est « : » :

.:~/java/mespackages:~/mesutil/cl.jar

- Sous Windows, le séparateur est « ; » :

.;c:\java\mespackages;c:\mesutil\cl.jar

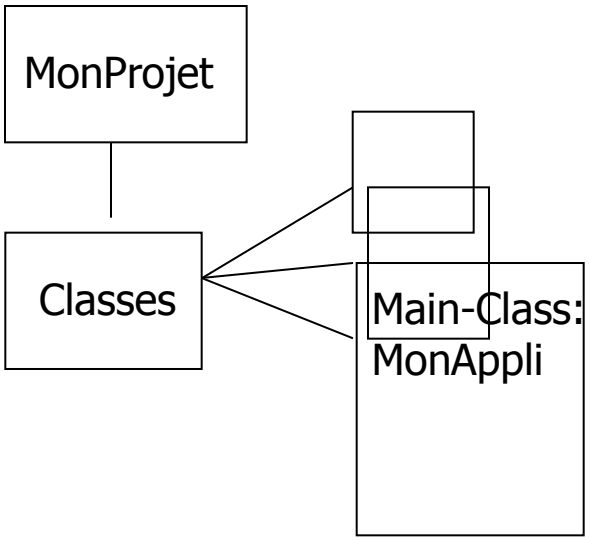
- Pour le développement d'applications complexes, il vaut mieux s'appuyer sur un utilitaire de type *make*
- L'utilitaire Ant, très évolué, et très utilisé par les développeur Java est spécialement adapté à Java
  - (http://www.club-java.com/Public/articles/antpaper.htm)

Placez votre code Java dans un fichier JAR

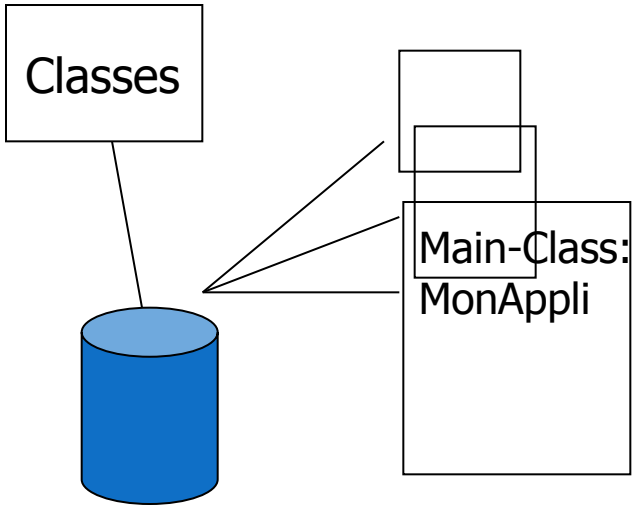
- JAR: Java Archive
  - Basé sur pkzip
  - Permet de grouper toutes les classes
  - Equivalent à la commande tar sous Linux
- Un fichier JAR est directement exécutable
  - Un utilisateur lance l'application sans extraire les fichiers

Création d'un JAR exécutable

- Création d'un fichier manifest.txt qui déclare dans quelle classe se trouve la méthode main():
- Déclaration:
  - Main-Class: MonAppli
- Création
  - Jar –cvmf manifest.txt app1.jar *.class
  - ou Jar –cvmf manifest.txt app1.jar MonAppli.class
- Pour exécuter: Java –jar app1.jar
- Remarque: le JAR ne contient pas de code source que du .java

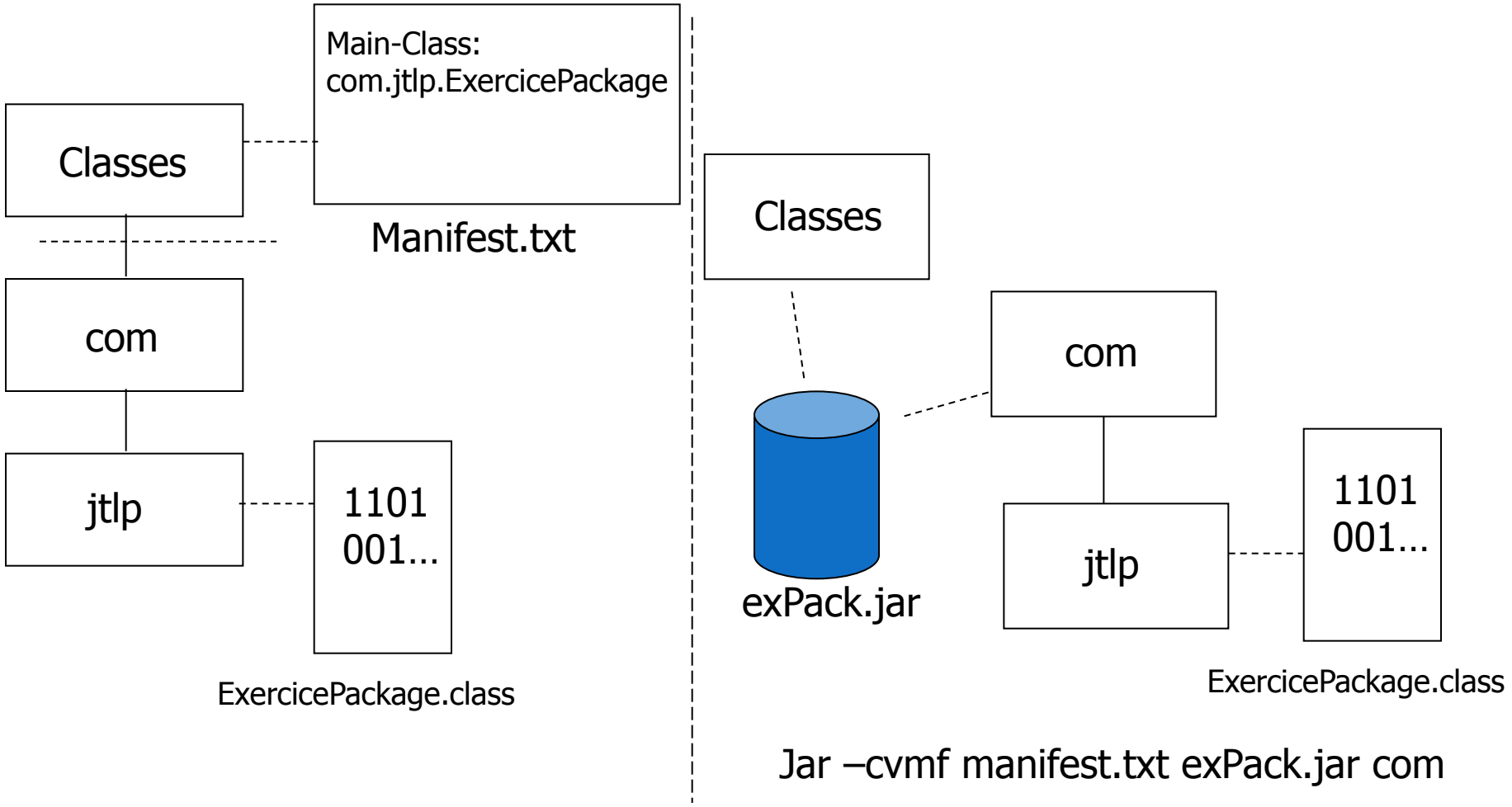


Manifest.txt



app1.jar

Créer le JAR exécutables avec des packages



- Lister le contenu d'un jar
  - Jar –tf exPack.jar
- Extraire le contenu d'un jar
  - Jar –xf exPack.jar