Haute École de Bruxelles
École Supérieure d'Informatique

TD Packages

Résumé

Ce TD a pour but d'aborder comment associer un package à vos méthodes.

1 Les package

1.1 Introduction aux packages

Vous êtes capables d'utiliser une classe qui a été placée dans un package standard (comme `java.util.Scanner`). Nous allons à présent vous montrer comment placer vos propres classes dans des packages et les utiliser.

Le nom d'un package

Un nom de package doit être choisi de telle sorte à représenter l'organisation à laquelle elle appartient et le projet associé ou le type de classe. Il ne faut pas se baser sur l'endroit où sont placés les fichiers sources.

Pour **votre** code, nous vous recommandons de rassembler vos classes par package reprenant votre login et le TD. Exemple : `g32010.tdPackage`.

Associer une classe à un package

L'appartenance d'une classe à un package déterminé est nécessaire à la compilation. Dès lors vous devrez ajouter comme **première instruction du source** (c-à-d avant même l'instruction `public class NomClasse`) :

```
package g32010.tdPackage;
```

Nom complet d'une classe

Le nom **qualifié** d'une classe (on dit aussi **nom complet**) est obtenu en accolant le nom de la classe au nom du package ; on obtient ainsi un nom unique pour cette classe. C'est ce nom qu'il faudra utiliser pour **exécuter** la classe.

Par exemple, le nom complet de la classe `Color` dont le package est `esi.util` est `esi.util.Color`.

Donnez le nom complet de la classe `SurfaceTriangle` dont le package est `g32010.tdPackage` : _____

Package et dossiers

Alors qu'on peut placer ses fichiers sources (les .java) où on veut, ce n'est pas le cas pour les fichiers compilés (les .class) dès lors qu'on joue avec des packages. Ils devront être placés à un endroit bien précis pour que le compilateur et la machine virtuelle puissent les retrouver.

La notion de package est liée à celle de répertoire (et même d'arborescence de répertoires). Ainsi le package `td.tdPackage` sera associé aux dossiers `td/tdPackage` (un dossier `tdPackage` dans un dossier `td`). Tout comme une classe appartient à un package, la version compilée de la classe devra se trouver dans les dossiers correspondant au package.

Exemple

Si `Color` a pour package `esi.util`, le fichier `Color.class` devra se trouver dans le répertoire associé `esi/util`.

Exercice

Examinez le contenu du dossier `esi/util` qui se trouve dans `/eCours/Java`.

Exercice

Donnez la suite de répertoires dans lesquels **doit** certainement se trouver la classe `SurfaceTriangle` dont le package est `g32010.tds.tdPackage`

1.2 Utiliser le package d'un autre

Une classe `Color` a comme package `esi.util`. Voyons comment l'utiliser.

Expérience

1. La classe possède une méthode principale. Tentez de l'exécuter via la commande `java Color`.
   Ca ne fonctionne pas. Pourquoi ?
2. Mais bien sûr ; on a dit qu'il fallait utiliser le nom complet de la classe. Tentez la commande `java esi.util.Color`.
   Zut ! Ca ne fonctionne toujours pas. Pourquoi ?
3. Parce que Java ne sait pas où trouver la classe et il est hors de question de chercher dans tout le système de fichier.
   Où est-elle d'ailleurs, cette classe ?
   Lançez la commande `find /eCours/java -name Color.class` pour le savoir. Vous devriez voir que la classe se trouve exactement ici : `/eCours/java/esi/util/Color.class`
4. On va indiquer à Java où chercher. Entrez la commande
   `java -cp /eCours/java esi.util.Color`.
   Cette fois-ci ça fonctionne !

L'option `cp` (une abréviation pour `classpath`) indique le (ou les) endroit(s) où chercher les classes.

Important ! on ne donne pas le dossier où se trouve le *.class* mais le dossier où il va pouvoir trouver la hiérarchie de dossiers liée au package. Finalement, le fichier se trouve à un endroit qui est la *concaténation* de l'option *cp* et du package.

1.3 Utiliser ses propres packages

À présent, nous allons voir comment vous pouvez placer vos propres classes dans un package.

Expérience

1. Prenez une copie de la classe `Color`.
2. Ajoutez comme première ligne :

   ```
   package g12345.util;
   ```

3. Compilez-la : `javac Color.java`.
4. Exécutez-la : `java -cp . g12345.util.Color`.
   Ça ne fonctionne pas ! Pourquoi ?
5. Qu'on est bête ! Java cherche le fichier dans une hiérarchie de dossiers correspondant au package. Ici, il cherche le fichier `g12345/util/Color.class`
6. Déplaçons le fichier au bon endroit.
   `mkdir -p g12345/util`
   `mv Color.class g12345/util`
7. Re-tentons : `java -cp . g12345.util.Color`.
   Ça fonctionne !

J'ai oublié l'option '-cp' et ça fonctionne quand même ! ?

En effet, sur linux1, et nous verrons pourquoi, si on ne lui dit pas où chercher, il cherche dans le dossier courant.

L'option -d

Ce serait pénible de devoir déplacer, après chaque compilation, la classe au bon endroit. Heureusement, le compilateur propose une option qui place directement le fichier généré au bon endroit.

La commande

```
java −d chemin Fichier.java
```

compile le fichier donné et place le résultat dans une hiérarchie de dossiers qui correspond au package à partir du chemin donné.

Exemple

Pour la classe couleur, on aurait pu compiler simplement avec la commande : `javac -d . Color.java` pour indiquer de créer le résultat dans le dossier `./g12345/util/`.

1.4 La variable CLASSPATH

Spécifier à chaque fois, l'option `cp` est pénible. Ce serait bien de pouvoir lui dire une fois pour toutes où chercher. C'est exactement le rôle de la variable d'environnement `CLASSPATH`.

La variable `CLASSPATH` contient une liste de dossiers où chercher les classes. Chaque dossier est séparé par `":"`.

Pour la manipuler, la procédure est la même que pour les autres variables d'environnement.

- Pour voir son contenu : `echo $CLASSPATH`
- Pour changer son contenu : `CLASSPATH=valeur`
- Pour ajouter un dossier à son contenu : `CLASSPATH=$CLASSPATH:valeur`
- Si elle est définie pour la première fois : `export CLASSPATH`
- Si la modification doit être permanente, vous pouvez placer la commande dans le fichier `.bashrc`

Exercice

Affichez le contenu actuel de la variable `CLASSPATH`. Que signifie le `"."` ?

Exercice

Sachant que la classe `SurfaceTriangle` se trouve dans `/home/g32010/tds/tdPackage` et qu'on retrouve `/home` dans la variable `CLASSPATH`, donnez l'instruction `package` correspondant à la situation :

```
package ____________________________________ ;
```

Exercice

Supposons que la classe `Exercice1` a pour package `esi.lg1.exemples` et qu'elle a comme chemin `/eCours/java/be/heb/esi/lg1/exemples/Exercice1.class`, donnez la hiérarchie de répertoires à ajouter au `CLASSPATH`.

CLASSPATH=$CLASSPATH : _____

Exercice

Faites ce qu'il faut pour pouvoir exécuter **notre** classe `Color` (package `esi.util`) sans utiliser l'option `-cp`.

1.5 Organiser ses fichiers

Résumons ce que nous avons déjà vu.

- un package est un regroupement de classes ;
- pour créer un tel package, il suffit de commencer les **fichiers sources** contenant les classes à regrouper par l'instruction `package` suivi du nom que l'on désire donner au package ;
- les **fichiers classes** doivent se trouver dans l'arborescence de répertoires donnée par le package.
- Cette arborescence doit commencer dans un dossier repris dans le CLASSPATH.

Mais, concrétement, quel dossier choisir pour placer les classes ? Il existe plusieurs façons de s'organiser ; on va vous en présenter deux.

Sans package.

D'abord, résumons ce que vous faisiez jusqu'à présent sans package. Avec cette approche, le source et la classe se trouvent dans un même dossier, quelconque.

1. On se place où on veut.
2. On édite le source : `nano Test.java`
3. On compile : `javac Test.java`
4. On exécute : `java Test`

1re proposition : pour transporter vos sources et classes rapidement.

Dans cette approche, les sources sont séparés des classes mais sont dans un dossier commun.

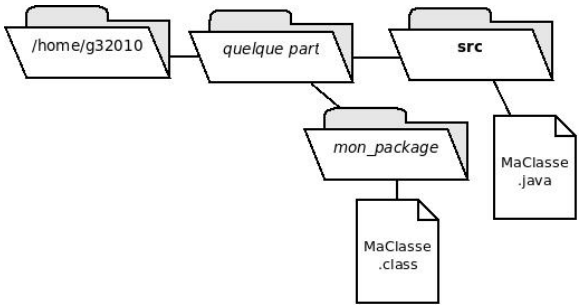

FIGURE 1 – Illustration de la 1ère approche

1. On vérifie que le CLASSPATH contient bien le dossier courant (sur linux1, c'est le cas).
2. On se place quelque part.
3. On crée un dossier pour les sources : `mkdir src`
4. On édite le source dans le dossier dédié :
   `nano src/Test.java`
   en prenant soin de commencer le fichier par un package qui a du sens (par exemple : `g12345.tdPackage`).
5. On compile en demandant à créer la classe à partir du dossier courant :
   `javac -d . src/Test.java`
6. On exécute : `java g12345.tdPackage.Test`

Exercice

Dans un sous-dossier du tdPackage (par exemple : tdPackage/cas1), faites une copie de votre programme `Hello.java` et placez-le dans un package en suivant l'approche ci-dessus. Quelle est la commande à utiliser pour compiler ? Et pour exécuter ?

Remarques

1. Il existe de nombreuses variantes. Par exemple, les sources dans le dossier "src" et les classes dans le dossier "classes" ou encore les classes

dans le dossier "classes" mais les sources directement dans le dossier courant.

2. Cette approche permet de facilement copier tous les sources et les classes associées
3. Par contre, les classes ne peuvent pas s'exécuter de partout.

2e proposition : toutes les classes sont regroupées dans le même dossier (~/classes).

Dans cette approche, toutes les classes de tous vos projets sont placées dans un dossier commun (par exemple : `~/classes`)

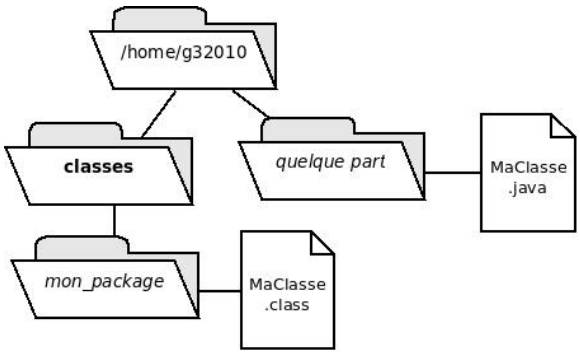

FIGURE 2 – Illustration de la 2ème approche

1. Il faut s'assurer que le CLASSPATH contienne le dossier `~/classes`. Si ce n'est pas le cas, il faut le définir (une fois pour toutes dans le fichier de configuration du bash) : `CLASSPATH=$CLASSPATH:~/classes`.
2. On se place quelque part.
3. On édite le source directement dans le dossier courant : `nano Test.java` en prenant soin de commencer le fichier par un package qui a du sens (par exemple : `g12345.tdPackage`).
4. On compile en demandant à créer la classe dans le dossier global dédié : `javac -d ~/classes Test.java`
5. On exécute : `java g12345.tdPackage.Test`

Exercice

Il s'agit du même exercice que pour la première approche. Dans un autre sous-dossier du tdPackage (par exemple : tdPackage/cas2), faites une copie de votre programme `Hello.java` et placez-le dans un package en suivant l'approche ci-dessus. Quelle est la commande à utiliser pour compiler ? Et pour exécuter ?

2 Exercices

Maintenant, mettons tout ça en pratique.

2.1 La notion de package

API

Ouvrez la documentation de l'API Java et recherchez la classe qui se nomme `IllegalFormatException`.

— Quel est le package de cette classe ? _____
— Quel est le nom qualifié de cette classe ?
_____

Écrivez la déclaration correcte d'une variable nommée `clavier` de type `Scanner` sans import de la classe.

_____ _____

Choix Multiple

Soit le code :

```
package td.tdPackage;
public class ErrCompilation {
    import java.util.Scanner;
    public static void main (String [] args) {
        System.out.println("TDPackage");
    }
}
```

la commande `javac ErrCompilation.java` provoque les erreurs suivantes :

```
ErrCompilation.java:2: illegal start of type
import java.util.Scanner;
^
ErrCompilation.java:2: ';' expected
import java.util.Scanner;
^
ErrCompilation.java:2: illegal start of type
import java.util.Scanner;
^
```

```
ErrCompilation.java:2: ';' expected
import java.util.Scanner;
^
ErrCompilation.java:2: <identifier> expected
import java.util.Scanner;
^
5 errors
```

Il s'agit d'erreurs générées par le compilateur car :

- `import` s'écrit avec une majuscule
- le package utilisé est incorrect
- le `import` est mal placé dans le code
- `Scanner` s'écrit avec une minuscule
- on n'utilise pas `Scanner` dans le code

Choix Multiple

Soit le code :

```
import java.util.Scanner;
package survol;
public class ErrCompilation{
    public static void main (String [] args){
        System.out.println("TDPackage");
    }
}
```

la commande `javac ErrCompilation.java` provoque l'erreur suivante :

```
ErrCompilation.java:3: class, interface, or enum expected
package survol;
^
1 error
```

Il s'agit d'une erreur générée par le compilateur `javac` car :

- `package` s'écrit avec une majuscule
- l'instruction `package` est mal placée dans le code
- le `package` utilisé est incorrect

Choix Multiple

Après correction du code précédent situé dans le répertoire survol, la suite de commandes :

`javac ErrCompilation.java`

`java ErrCompilation`

provoque l'erreur suivante :

```
Exception
in thread "main" java.lang.NoClassDefFoundError:
ErrCompilation (wrong name: survol/ErrCompilation)
at java.lang.ClassLoader.defineClass1(Native Method)
at java.lang.ClassLoader.defineClass(ClassLoader.java:632)
at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:142)
at java.net.URLClassLoader.defineClass(URLClassLoader.java:277)
at java.net.URLClassLoader.access$000(URLClassLoader.java:73)
at java.net.URLClassLoader$1.run(URLClassLoader.java:212)
at java.security.AccessController.doPrivileged(Native Method)
at java.net.URLClassLoader.findClass(URLClassLoader.java:205)
at java.lang.ClassLoader.loadClass(ClassLoader.java:319)
at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:294)
at java.lang.ClassLoader.loadClass(ClassLoader.java:264)
at java.lang.ClassLoader.loadClassInternal(ClassLoader.java:332)
Could not find the main class: ErrCompilation. Program will exit.
```

Il s'agit d'une erreur générée par la machine virtuelle Java car :

- ❍ pour exécuter on doit donner le nom qualifié de la classe
- ❍ la déclarative de package ne correspond pas à l'emplacement du fichier .class
- ❍ la machine virtuelle n'est pas configurée pour reconnaitre l'utilisation de packages
- ❍ le fichier .class devrait se trouver ailleurs que le source Java

2.2 À vous de jouer...

Voici quelques conseils pour vous guider dans la résolution de tels problèmes :

- — il convient d'abord de bien comprendre le problème posé ; assurez-vous qu'il est parfaitement spécifié ;
- — résolvez le problème via quelques exemples précis ;
- — mettez en évidence les variables «**données** », les variables «**résultats** » et les variables de travail ;
- — n'hésitez pas à faire une ébauche de résolution en français avant d'élaborer l'algorithme définitif pseudo-codé ;
- — déclarez ensuite les variables (et leur type) qui interviennent dans chaque algorithme ; les noms des variables risquant de ne pas être suffisamment explicites.
- — Écrivez la partie algorithmique **AVANT** de vous lancer dans la programmation en Java.
- — Pour la partie Java, dessinez l'arborescence des fichiers.

Vous trouverez cet énoncé ici (www.heb.be/esi/TDPackage/fr/../../TDPackage/fr/enonce/DEV1-LAJ-interro-3B-Java-JLC.pdf).