Les packages

Un problème courant dans les langages de programmation est celui de la collision de noms. Si par exemple, je veux créer une classe *TextEditor* pour représenter une composant graphique complexe pour éditer un texte, un autre développeur peut également le faire. Si nous distribuons nos classes, cela signifie qu'une application peut se retrouver avec deux classes dans son *classpath* qui portent exactement le même nom mais qui ont des méthodes et des comportements différents.

Dans la pratique, la JVM chargera la première classe qu'elle peut trouver et ignorera la seconde. Ce comportement n'est pas acceptable. Pour cela, il faut pouvoir différencier ma classe *TextEditor* d'une autre.

Le moyen le plus efficace est d'introduire un espace de noms qui me soit réservé. Dans cet espace, *TextEditor* ne désignerait que ma classe. En Java, les **packages** servent à délimiter des espaces de noms.

Déclaration d'un package

Pour qu'une classe appartienne à un package, il faut que son fichier source commence par l'instruction :

```
package [nom du package];
```

Une classe ne peut appartenir qu'à un seul package. Les packages sont également représentés sur le disque par des répertoires. Donc pour la classe suivante :

```
package monapplication;


public class TextEditor {


}
```

Cette classe doit se trouver dans le fichier *TextEditor.java* et ce fichier doit lui-même se trouver dans un répertoire nommé *monapplication*. Pour les fichiers *class* résultants de la compilation, l'organisation des répertoires doit être conservée (c'est d'ailleurs ce que fait le compilateur). Ainsi, si deux classes portent le même nom, elles se trouveront chacune dans un fichier avec le même nom mais dans des répertoires différents puisque ces classes appartiendront à des packages différents.

> Quand on spécifie le**classpath** à la compilation ou au lancement d'un programme, on spécifie le ou les répertoires à partir desquels se trouvent les packages.

Si une classe ne déclare pas d'instruction **package** au début du fichier, on dit qu'elle appartient au package par défaut (qui n'a pas de nom). Même si le langage l'autorise, c'est quasiment toujours une mauvaise idée. Les IDE comme Eclipse signalent d'ailleurs un avertissement si vous voulez créer une classe dans le package par défaut. Jusqu'à présent, les exemples donnés ne mentionnaient pas de package. Mais maintenant que cette notion a été introduite, les exemples à venir préciseront toujours un package.

Sous package

Comme pour les répertoires, les packages suivent une organisation arborescente. Un package contenu dans un autre package est appelé un **sous package** :

```
package monapplication.monsouspackage;
```

Sur le système de fichiers, on trouvera donc un répertoire *monapplication* avec à l'intérieur un sous répertoire *monsouspackage*.

Nom d'un package

Comme le mécanisme des packages a été introduit pour éviter la collision de noms, il est conseillé de suivre une convention de nommage de ses packages. Pour une organisation, on utilise le nom de domaine inversé comme base de l'arborescence de packages : par exemple com.cgi.udev. On ajoute généralement ensuite le nom de l'application ou de la bibliothèque.

Les noms de packages contenant le mot *java* sont réservés pour la bibliothèque standard. On trouve ainsi des packages *java* ou *javax* (pour indiquer une extension de Java) dans la bibliothèque standard fournie avec le JDK.

Nom complet d'une classe

Une classe est normalement désignée par son *nom complet*, c'est-à-dire par le chemin de packages suivi d'un **.** suivi du nom de la classe.

Par exemple, la classe String s'appelle en fait java.lang.String car elle se trouve dans le package java.lang. J'ai donc la possibilité, si je le souhaite, de créer ma propre classe String par exemple dans le package com.cgi.udev :

```
package com.cgi.udev;

public class String {

}
```

Il est possible d'accèder à une classe en spécifiant son nom complet. Par exemple, pour accèder à la classe java.util.Arrays :

```
package com.cgi.udev;

public class MaClasse {

  public static final main(String... args) {
    int[] tableau = {5, 6, 3, 4};
    java.util.Arrays.sort(tableau);
  }
}
```

Par défaut, une classe a accès à l'espace de nom de son propre package et du package java.lang. Voilà pourquoi, il est possible d'utiliser directement les classes String ou Math sans avoir à donner leur nom complet : java.lang.String, java.lang.Math.

Si nous créons deux classes : *Voiture* et *Conducteur*, toutes deux dans le package com.cgi.udev :

```
package com.cgi.udev;

public class Conducteur {

  // ...

}
```

```
package com.cgi.udev;

public class Voiture {
```

```
  private Conducteur conducteur;

  public void setConducteur(Conducteur conducteur) {
    this.conducteur = conducteur;
  }

  // ...

}
```

La classe *Voiture* et la classe *Conducteur* appartiennent toutes les deux au package com.cgi.udev. La classe *Voiture* peut donc référencer la classe *Conducteur* sans préciser le package.

Import de noms

Pour éviter de préfixer systématiquement une classe par son nom de package, il est possible d'importer son nom dans l'espace de noms courant grâce au mot-clé **import**. Une instruction **import doit** se situer juste après la déclaration de **package** (si cette dernière est présente). Donc, il n'est pas possible d'importer un nom en cours de déclaration d'une classe ou d'une méthode.

Le mot-clé **import** permet d'importer :

- Un nom de classe particulier

```
import java.util.Arrays;
```

- Un nom de méthode de classe ou d'attribut de classe

```
import static java.lang.Math.abs;
import static java.lang.System.out;
```

- Un nom de classe interne (inner class)

```
import java.util.Map.Entry;
```

- Tous les noms d'un package

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

- Tous les noms des méthodes et des attributs de classe

```
import static java.lang.Math.*;
```

Le caractère * permet d'importer tous les noms d'un package dans l'espace de nom courant. Même si cela peut sembler très pratique, il est pourtant déconseillé de le faire. Tous les IDE Java savent gérer automatiquement les importations. Dans Eclipse, lorsque l'on saisit le nom d'une classe qui ne fait pas partie de l'espace de nom, il suffit de demander la complétion de code (*CTRL + espace*) et de choisir dans la liste la classe appartenant au package voulu et Eclipse génère automatiquement l'instruction **import** pour ce nom de classe. De plus, on peut demander à Eclipse à tout moment de réorganiser les importations (*CTRL + MAJ + O*). Ainsi, la gestion des importations est grandement automatisée et le recours à * comme facilité d'écriture n'est plus vraiment utile.

```
package com.cgi.udev;

import static java.lang.Math.random;
import static java.lang.System.out;
import static java.util.Arrays.sort;

import java.time.Duration;
import java.time.Instant;

public class BenchmarkTriTableau {

  public static void main(String[] args) {
    int[] tableau = new int[1_000_000];

    for (int i = 0; i < tableau.length; ++i) {
      tableau[i] = (int) random();
    }

    Instant start = Instant.now();
    sort(tableau);
    Duration duration = Duration.between(start, Instant.now());

    out.println("Durée de l'opération de tri du tableau : " + duration);
  }

}
```

> Si vous importez un nom qui est déjà défini dans l'espace courant, alors l'import n'aura aucun effet. Dans ce cas, vous serez obligé d'accéder à un nom de classe avec son nom long afin d'éviter toute ambiguïté.

La portée de niveau package

Nous avons vu précédemment que les classes, les méthodes et les attributs peuvent avoir une portée **public** ou **private**. Il existe également une portée de niveau package. Une classe, une méthode ou un attribut avec cette portée n'est accessible qu'aux membres du même package. Cela permet notamment de créer des classes nécessaires au fonctionnement du package tout en les dissimulant aux éléments qui ne sont pas membres du package.

Il n'y a pas de mot-clé pour désigner la portée de niveau package. Il suffit simplement d'omettre l'information de portée.

Imaginons que nous voulions créer une bibliothèque de cryptographie. Nous pouvons créer une classe pour chaque algorithme. Par contre, pour simplifier l'utilisation, nous pouvons fournir une classe outil de chiffrement. Dans ce cas, il n'est pas nécessaire de rendre accessible à l'extérieur du package les classes représentant les algorithmes : on les déclare alors avec la portée package.

CypherAlgorithm.java¶

```
package com.cgi.udev.cypher;

class CypherAlgorithm {

  public CypherAlgorithm() {
    // ...
  }

  public byte[] encrypt(byte[] msg) {
    // ...
  }
}
```

CypherLibrary.java¶

```
package com.cgi.udev.cypher;

public class CypherLibrary {
```

```
  private CypherLibrary() {
  }

  public static byte[] cypher(byte[] msg) {
    CypherAlgorithm algo = new CypherAlgorithm();
    return algo.cypher(msg);
  }
}
```

La classe *CypherAlgorithm* est de portée package, elle est donc invisible pour les classes qui ne sont pas membres de son package. Par contre, elle est utilisée par la classe *CypherLibrary*.

La portée de niveau package est souvent utilisée pour dissimuler la complexité de l'implémentation en ne laissant voir que les classes et/ou les méthodes réellement utiles aux utilisateurs.

Le fichier package-info.java

Il est possible de créer un fichier spécial dans un package nommé *package-info.java*. Au minimum, ce fichier doit contenir une instruction **package**. Ce fichier particulier permet d'ajouter un commentaire Javadoc pour documenter le package lui-même. Il peut également contenir des annotations pour le package.

contenu du fichier package-info.java pour com.cgi.udev¶

```
package com.cgi.udev;


/**
 * Ceci est le commentaire pour le package.
 */
```

Revision #1
Created 10 February 2025 11:11:10 by Nicolas
Updated 10 February 2025 11:13:14 by Nicolas