CENTRALE NANTES

Programmation Orientée Objet - Les paquetages en Java

cours de Jean-Marie Normand

Présentation - Définition

- En Java, les classes sont regroupées en paquetages (*packages* en anglais)
- Les paquetages de part leur utilisation peuvent être vus comme les bibliothèques d'autres langages (par exemple C/C++, etc.)
- Un paquetage regroupe des classes qui ont une relation sémantique, qui traitent d'un même sujet, etc.
- Le JDK contient énormément de paquetages à référez vous à la documentation en ligne

CENTRALE NANTES

Quelques paquetages du JDK

- `java.lang` : classes de base du langage Java
- (`String`, `Math`, etc.)
- `java.util` : ensemble d'utilitaires du langage Java, comme par exemple les conteneurs (`ArrayList`, `List`, etc.)
- `java.io` : classes qui permettent de réaliser des entrées/sorties (par exemple de lire et d'écrire dans des flux ou fichiers)
- `java.sql` : classes permettant de lancer des `requêtes SQL` et de gérer des `connections` entre un programme Java et des `bases de données` (connecteur JDBC – Java DataBase Connector)
- `java.swing` : classes permettant de réaliser des interfaces graphiques

Intérêt des paquetages

- Facilitent la modularité par une meilleure organisation du code (regroupement de classes sémantiquement proches, organisation au sein de dossiers, etc.)
- Eviter des conflits de noms (plusieurs classes ayant le même nom par exemple)
- Meilleure gestion des droits d'accès aux attributs et aux méthodes au sein d'un paquetage (cf. CM Chapitre 2)

Paquetage et nom complet d'une classe en Java

- En Java, le nom complet d'une classe contient le nom du paquetage à laquelle elle appartient :

  `java.util.List`

- Il est ainsi possible de distinguer deux classes qui ont le même nom mais n'appartiennent pas au même paquetage :

  `java.util.List` *(liste d'objets)*

  `java.awt.List` *(liste déroulante dans une interface utilisateur)*

- Une classe du même paquetage peut être désignée par son nom « terminal » :

  Les classes du paquetage `java.awt` *peuvent désigner directement la liste déroulante en utilisant* `List`

- Une classe d'un autre paquetage doit être désignée par son nom complet, avec le nom du paquetage complet en préfixe :

  On utilise `java.awt.List` *pour désigner la classe en dehors du paquetage* `java.awt`

> **Problème : ça peut devenir très très long !**
> **Solution : Java permet d'importer des classes ET des paquetages !**

Importation d'une classe

- Pour pouvoir désigner une classe par son nom « terminal », il faut l'importer dans son code Java :

> Utilisation de l'instruction `import`

- Exemple en début de fichier `Exemple.java` :

```
> import java.util.ArrayList;
```

- Dans le reste du fichier `Exemple.java`, on pourra directement utiliser `ArrayList` au lieu de `java.util.ArrayList`
- NB : il est tout à fait possible d'utiliser une classe sans l'importer !

Exemple sans import

```java
/*
 * Fichier Exemple.java
 */

/**
 *
 * @author Jean-Marie Normand <jean-marie.normand@ec-nantes.fr>
 */
public class Exemple {
    // début de la classe
    java.util.ArrayList<Integer> maListe = new java.util.ArrayList<>();
    // vous comprendrez ce code un peu plus tard
}
```

Exemple avec import

```java
/*
 * Fichier Exemple.java
 */

/**
 *
 * @author Jean-Marie Normand <jean-marie.normand@ec-nantes.fr>
 */

import java.util.ArrayList;

public class Exemple {
    // début de la classe
    ArrayList<Integer> maListe = new ArrayList<>();
    // vous comprendrez ce code un peu plus tard
}
```

Importer TOUTES les classes d'un paquetage à la fois

- Java nous permet également d'importer TOUTES les classes d'un paquetage en même temps :

  > *utilisation du joker : '*'*
  > *`import java.util.*;`*

  > *Importe toutes les classes du paquetage `java.util` !*
  > *On peut donc les utiliser directement dans notre code !*

Lever une ambiguïté

- Il est possible d'arriver à des situations ambiguës où l'on importe plusieurs paquetages qui comportent des classes de même nom → le compilateur Java génère alors une erreur
- Il est possible d'importer plusieurs paquetages contenant des classes homonymes → mais il faudra OBLIGATOIREMENT utiliser le nom complet de ces classes !

CENTRALE NANTES

Exemple d'ambiguïté

```
/*
 * Fichier Exemple.java
 */

/**
 *
 * @author Jean-Marie Normand <jean-marie.normand@ec-nantes.fr>
 */

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

reference to List is ambiguous, both class java.awt.List in java.awt and interface java.util.List in java.util match

reference to List is ambiguous, both class java.awt.List in java.awt and interface java.util.List in java.util match

---

(Alt-Enter shows hints)

```
    List maListe = new List<>();
    // vous comprendrez ce code un peu plus tard
}
```

Exemple d'ambiguïté

```java
/*
 * Fichier Exemple.java
 */

/**
 *
 * @author Jean-Marie Normand <jean-marie.normand@ec-nantes.fr>
 */

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



public class Exemple {
    // début de la classe
    java.awt.List maListe = new java.awt.List();
    // vous comprendrez ce code un peu plus tard
}
```

Remarque importante

- Par défaut, Java importe directement pour vous le paquetage :

> `java.lang`

- Il est inutile de l'inclure par vous même
- Vous pourrez donc utiliser directement dans vos programmes toutes les classes de ce paquetage

Mettre une classe dans un paquetage – Déclarer un paquetage

- Chaque paquetage possède un nom, qui est soit :
  - > *un identificateur unique (p. ex. `monPaquetage`)*
  - > *une suite d'identificateurs séparés par des points (p. ex. `coursObjet.monPaquetage`)*
- Vous pouvez indiquer qu'une classe fait partie d'un paquetage en ajoutant l'instruction suivante en début de fichier :
  - > `package nomCompletDeMonPaquetage;`
- Elle doit figurer au tout début des fichiers .java, avant même d'éventuelles instructions import !
- Les paquetages sont créés dynamiquement en
  fonction des déclarations faites dans les différentes classes (si le paquetage n'existait pas, il est créé dès que cette instruction est ajoutée à un fichier .java)

Mettre une classe dans un paquetage – Déclarer un paquetage

- Exemple de déclaration d'une classe comme membre d'un paquetage :
  > On définit ici la classe `Exemple` comme faisant partie du paquetage `monpaquetage`

```java
/*
 * Fichier Exemple.java
 */

/**
 *
 * @author Jean-Marie Normand <jean-marie.normand@ec-nantes.fr>
 */

package monpaquetage;

public class Exemple {
    // contenu de la classe
}
```

Paquetage par défaut

- **Attention** : en Java si vous ne spécifiez pas de nom de paquetage avant le nom de votre classe → Java inclut votre classe dans le paquetage par défaut !
- Si vous définissez plusieurs classes dans un répertoire sans spécifier de nom de paquetage à elles seront toutes dans le paquetage par défaut !
- Ceci a un impact important sur les droits d'accès de vos attributs et méthodes ! (cf. *CM Chapitre 2)*

Paquetage par défaut (2)

- Ici la classe `Exemple` appartient donc au paquetage par défaut de Java !

```java
/*
 * Fichier Exemple.java
 */

/**
 *
 * @author Jean-Marie Normand <jean-marie.normand@ec-nantes.fr>
 */
public class Exemple {
    // contenu de la classe

}
```

Hiérarchie de paquetages – Sous-paquetages

- Un paquetage peut avoir des sous-paquetages !

  > *`java.awt.event` est un sous-paquetage de `java.awt`*
  > *Il peut y avoir une hiérarchie de paquetages, sous-paquetages, etc.*

- Attention : l'import des classes d'un paquetage **n'importe pas** les classes du sous-paquetages !
- On doit donc écrire :

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

Convention de nommage d'un paquetage

- Il est conseillé de préfixer ses propres paquetages par ordre de généralité des paquetages :

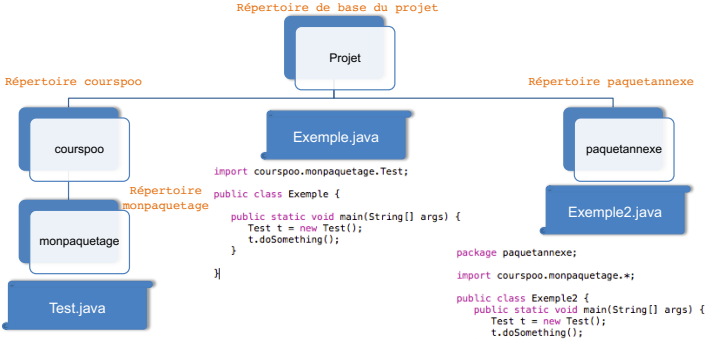
> `package java.awt.event;`

- On commence par indiquer le plus général : c'est la librairie de base java
- Ensuite plus spécifique : c'est du `awt` dans la librairie `java`
- Puis encore plus spécifique : c'est la partie qui gère les événements dans `awt`

> Autre exemple : `org.centrale.objet.projet`

- Projet du cours d'Objet de Centrale (Nantes) vue comme une organisation
- Permet d'éviter les ambiguïtés de noms de paquetages et de classes réalisées par d'autres programmeurs

Exemple de paquetages




```java
import courspoo.monpaquetage.Test;

public class Exemple {

   public static void main(String[] args) {
      Test t = new Test();
      t.doSomething();
   }

}
```

```java
package paquetannexe;

import courspoo.monpaquetage.*;

public class Exemple2 {
   public static void main(String[] args) {
      Test t = new Test();
      t.doSomething();
   }
}
```

```java
package courspoo.monpaquetage;

public class Test {
   public Test() {

   }

   public void doSomething() {
      System.out.println("Salut je suis dans monpaquetage !");
   }
}
```

Composants publiques d'un paquetage – Encapsulation dans un paquetage

- Lorsqu'une classe appartient à un paquetage :

  > Si la classe est publique (p. ex. `public class Exemple`) alors elle est visible et accessible de partout (dans et en dehors du paquetage)

```java
/*
 * Fichier Exemple.java
 */

/**
 *
 * @author Jean-Marie Normand <jean-marie.normand@ec-nantes.fr>
 */

package monpaquetage;

public class Exemple {
    // contenu de la classe
}
```

Composants publiques d'un paquetage – Encapsulation dans un paquetage

- Lorsqu'une classe appartient à un paquetage :

  > Sans modificateur de droit, elle est visible et accessible uniquement à l'intérieur du paquetage

```java
/*
 * Fichier Exemple.java
 */

/**
 *
 * @author Jean-Marie Normand <jean-marie.normand@ec-nantes.fr>
 */
package monpaquetage;

class Exemple {
    // contenu de la classe
}
```

Regroupement de classes au sein d'un paquetage physique : Archive Java (JAR)

- Avec le développement d'applications distribuées et/ou en ligne, il est nécessaire de s'assurer que :
  > *l'application puisse fonctionner,*
  > *tous les éléments nécessaires (packages, etc.) soient présents,*
  > *l'application prenne le moins d'espace possible*
- Pour répondre à ces contraintes, Java propose l'utilitaire `jar` dans le JDK
- Il permet de rassembler les différents classes compilées (fichiers d'extension .class) d'une application au sein d'une archive compressée

CENTRALE NANTES

Java Archive : JAR

- Création d'une archive .jar :

  > *`jar -cvf montest.jar montest/*`*
  > *Regroupe dans `montest.jar` tout le contenu du répertoire `montest/`*
  > *Regarder la documentation de jar pour voir toutes les options*

- On peut ensuite regrouper toutes les jar dans un même répertoire pour faciliter la compilation en ligne de commande

Option de compilation –classpath et variable d'environnement CLASSPATH

- On peut indiquer au compilateur comment accéder au répertoire contenant les archives .jar lorsque l'on veut compiler des fichiers java utilisant des packages .jar :

  > *javac -classpath monchemin/ monFic.java*

- Avec l'option de compilation –classpath
- Ou en modifiant la variable d'environnement CLASSPATH, par exemple sous Windows (dans le terminal) :

  > *set CLASSPATH=%CLASSPATH%;monchemin/*

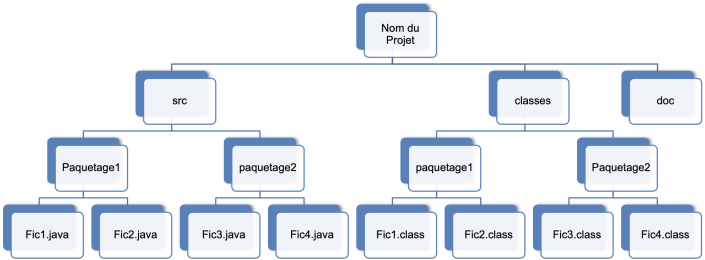
Utilisez des IDE (Environnement de Développement Intégré)

- La gestion des packages est beaucoup plus simple si on utilise un IDE comme :
  > *Eclipse*,
  > *NetBeans*,
  > *Etc.*

Structure classique d'un projet Java

- En général, il est préconisé d'organiser les fichiers d'un projet Java afin de ne pas tout mélanger
- Cette organisation est globalement la même pour tout le monde (si le projet est fait « à la main » ou via un IDE

> Même si de petites différences de nommage existent



NetBeans

- Lors de la création d'un nouveau projet :

> File à New Projet à Donnez un nom

- NetBeans créé automatiquement :

  > *un package du même nom que le projet*
  > *un répertoire « Libraries »*

- Pour ajouter des .jar à votre projet, il suffit de :

  > *Cliquer droit sur le répertoire Libraries → Add JAR/Folder*
  > *Choisir un `fichier.jar` ou le répertoire contenant les .jar*