Programmation orientée objets

Leçon 5 : Paquetages

Prof. Rolf Ingold

Département d'informatique

UNI
FR

UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Contenu de la leçon

- Rôle des paquetages
- Règles de nommage
- Clauses d'importation
- Règles de visibilité
- Paquetages de base de JDK

- Solution du challenge de la leçon 4

Unités de compilation (rappel)

- En java, les classes sont des unités de compilation
  - elle sont décrites dans un fichier qui porte le même nom et l'extension `.java`
  - le résultat de la compilation produit des fichiers avec le même nom et l'extension `.class`
  - tant qu'une classe n'est pas modifiée, il n'est pas nécessaire de la recompiler

- Les classes sont organisées en **paquetages**

- Pour la compilation et l'exécution, l'emplacement des paquetages doit être spécifié dans la variable système `CLASSPATH`

Rôle des paquetages

- Les **paquetages** constituent des regroupements logiques de classes (et d'interfaces) permettant de
  - regrouper (structurer) l'ensemble des classes de manière logique
  - résoudre les éventuels conflits de noms
  - contrôler plus finement l'accès aux membres d'une classe, grâce au mot-clé `protected`

Utilisation des paquetages

- Les paquetages servent à structurer les logiciels
  - Ils forment un ensemble cohérent de classes
- Les paquetages sont typiquement utilisés pour
  - constituer des librairies
  - répartir les tâches de programmation entre plusieurs programmeurs ou plusieurs équipes de programmeurs
- Les noms de paquetage sont hierarchisés

- Attention : la structure des paquetages est découplée de l'organisation hiérachique des classes
  - une sous-classe peut faire partie d'un paquetage totalement indépendant de celui de sa superclasse

Nommage des paquetages

- Les **paquetages** sont nommés au moyen d'un identificateur ou d'une cascade d'identificateurs spéarés par des points
  - exemples de noms de paquetages:

```
mypackage
java.util
java.util.zip
javax.swing
org.w3c.dom
oop.graphics
oop.demo.studies
```

  - par convention, les identificateurs sont toujours en minuscules (ils peuvent contenir le caractère _ et des chiffres, sauf au début)
- Malgré l'apparence, il n'y a pas de liens hiéarchiques entre les paquetages (sauf la structure des répertoires)
- Il existe un **paquetage par défaut**, anonyme, qui regroupe toutes les classes qui n'ont pas été assignées à un paquetage

Assignation d'une classe à un paquetage

- Pour assigner une classe à un paquetage, il faut ajouter tout au début du fichier une clause de paquetage de la forme

```
package myLib.example;
```

- Ensuite, il faut placer le fichier dans un répertoire correspondant, c'est-à-dire dans un répertoire dont le chemin d'accès est

```
src/myLib/example/
```

où `src` est le répertoire racine des programmes sources

Noms complets

- Pour utiliser une classe appartenant à un paquetage ou un de ses membres, on peut utiliser un **nom complet** (i.e. préfixé par le nom du paquetage)

```
new oop.graphics.Circle()
```

- Cette notation est utile pour lever les éventuels conflits de noms

Clauses d'imporation

- L'utilisation la plus fréquente consiste à déclarer une liste de **clauses d'importation**

```
import oop.lib.graphics.Circle;
import oop.lib.graphics.Square;
```

- Il est possible d'importer toutes les classes d'un paquetage avec l'abréviation `*`

```
import oop.lib.graphics.*;
```

  - mais c'est déconseillé ! (pour des raisons de documentation)

- Les classes du même paquetage sont importées d'office
- Les classes du paquetage `java.lang` sont aussi importées d'office

Importations statiques

- Dans la liste d'importation on peut également importer des membres statiques d'une classe

```
import static java.lang.Math.PI;
import static java.lang.Math.cos;
```

- Cela présente l'avantage de ne pas devoir préfxier le membre statique avec le nom de la classe

```
double x = cos(theta / 180.0 * PI)
```

au lieu de

```
double x = Math.cos(theta / 180.0 * Math.PI)
```

Attributs de visibilité définitives

- Nous avions déjà présenté les attributs `public` et `private`
  - il existe un troisième attribut : `protected`
  - enfin, il peut ne pas y avoir d'attribut

- Récapitulation des règles :
  - les membres ayant l'attribut `public` sont visibles partout
  - les membres ayant l'attribut `private` sont visibles seulement à l'intérieure de la classe
  - les membres sans attribut de visibilité sont seulement visibles
    - dans les classes du paquetage
  - les membres ayant l'attribut `protected` sont visibles
    - dans les classes du paquetage
    - et dans toutes les sous-classes

Java Development Kit

- Le **Java Development Kit** est livré avec une librairie constituée de plus de 200 paquetages contenant plusieurs milliers de classes et interfaces dont :
  - `java.lang` : classes fondamentales liées au langage Java (ne doit pas être importée)
  - `java.util` : classes d'utilité générale (ensembles, listes, tables, générateurs aléatoires, etc)
  - `java.io` : pour les opérations d'entrée-sorties
  - `java.sql` : pour l'accès aux bases de données relationnelles
  - `java.util.concurrent` : pour la programmation concurrente
  - `java.util.function` : pour la programmation fonctionnelle
  - `java.time` : pour la manipulation du temps (date et heure)
  - `java.text` : pour le formatage de nombres, de dates, etc.
  - `java.awt`, `java.awt.event` et `javax.swing` : pour la programmation d'applications interactives
  - `javafx.scene`, `javafx.geometry`, `javafx.event`, etc. : nouvelle bibliothèque pour les applications interactives
  - `org.w3c.dom` : pour la manipulation de douments XML
  - ....

UNI FR
UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Complément : Challenge de la leçon 4

- Rappel de la problématique
- Solution proposée
- Démonstration

Rappel de la problématique

- La classe **`Polygon`** permet d'accéder aux points et, en les déplaçant, de modifier la forme
  - pour un polygone, il n'y a rien de gênant; c'est une fonctionnalité souhaitée
  - pour un parallélogramme, un rectangle ou un carré, cela pose problème puisqu'en déplaçant un seul point on détruit sa supposée propriété !

- Comment peut-on palier ce problème ?

Solution proposée : classe `SmartPoint`

- Il s'agit d'empêcher le déplacement de certains points (par ex. les sommets d'un rectangle)

- Le principe consiste à permettre à étendre la class **`Point`** de manière à pouvoir "geler" les points temporairement

- L'idée est de créer une sous-classe **`SmartPoint`** dérivée de **`Point`** qui contient
  - un attribut **`frozen`**
  - des méthodes **`freeze`** et **`unfreeze`**
  - une redéfinition des méthodes **`translate(...)`**, **`rotate(...)`** et **`scale(...)`** qui vérifie si le point n'est pas "gelé" et sinon, envoie un message d'erreur

Solution proposée : classe `SmartPolygon`

- Pour tirer profit de la fonctionnalités de **`SmartPoint`**, il faut adapter les classes d'objets graphiques à protéger : **`Rectangle`**, **`Triangle.Equilateral`**, etc.

- Concrètement, il s'agit de
  - affecter aux sommets des points de la classe **`SmartPoint`**
  - geler ces points lors de la création
  - redéfinir des méthodes **`translate(...)`**, **`rotate(...)`** et **`scale(...)`** afin de permettre le déplacement des points

- Pour éviter la duplication de code, il convient de factoriser ces adaptations dans une classe **`SmartPolygon`** qui étend la classe **`Polygon`**
  - ainsi, il suffit de modifier la clause **`extend`** des classes d'objets graphiques à protéger

Solution proposée : attributs de visibilité

- Pour que ces modifications soient transparentes du point de vue des programmes externes au paquetage, il est recommandé de limiter la visibilté
  - les classes **`SmartPoint`** et **`SmartPolygon`** sont non publiques (donc seulement visible dans le paquetage **`xxx.geom`**
  - dans **`SmartPoint`**, les méthodes **`freeze()`** et **`unfreeze()`** sont déclarées **`protected`**

- Conclusions
  - les modifications proposées sont totalement transparentes pour les classes clientes du paquetage
  - ces classes n'ont pas la possibilité d'interférer avec l'état frozen ou non des points