Travaux Dirigés de Java n°5

— Etienne.Duris@univ-mlv.fr —

▸ Exercice 1. (Qui suis-je ?)

Écrire un petit programme de test qui permet d'obtenir et d'afficher la représentation sous forme `String` *les objets "classe" (de type* `java.lang.Class`*) correspondant :*

- *à une instance* `s` *de la classe* `String`*;*
- *au nom de la classe* `String`*;*
- *à une instance* `i` *de la classe* `Integer`*;*
- *à un entier* `i` *de type* `int`*;*
- *à une instance de la classe* `Vector`*;*
- *au nom de la classe* `Vector`*;*
- *au nom de la classe* `Personne` *définie au TD 1 ;*
- *au nom de l'interface* `Cloneable`*;*
- *au nom de l'interface* `Evaluable` *définie au TD 3.*

▸ Exercice 2. (Introspection)

À partir du nom de la classe de l'API standard permettant de manipuler les dates, `java.util.Date`*, récupérer l'objet de classe* `Class` *correspondant (votre programe ne doit pas faire de* `import java.util.Date`*).*

Récupérer, à partir de cet objet classe :

- *l'objet méthode, de classe* `java.lang.reflect.Method`*, correspondant à la méthode de nom* `toString` *qui n'accepte aucun argument ;*
- *l'objet constructeur, de classe* `java.lang.reflect.Constructor`*, défini dans la classe* `Date` *avec un paramètre de type* `long`*;*

Utiliser ces objets pour :

1. *Créer une instance de la classe* `Date` *à partir de l'objet classe correspondant et afficher cette instance en utilisant la méthode récupérée par introspection.*

2. *Créer une instance de la classe* `Date` *à partir du constructeur précédement récupéré par introspection et appeller ce constructeur avec un entier long de valeur* `0`. *Afficher l'objet ainsi crée avec la méthode récupérée par introspection.*

▸ **Exercice 3. (Programmation Dynamique)**

Dans cet exercice, on veut écrire un programme qui soit capable, dynamiquement (à l'exécution), d'utiliser différentes ressources. Dans notre exemple, ces ressources sont des classes de manipulations de formats et ont pour point commun d'implanter la même interface `Format`.

Lorsque une ou plusieurs classes implantant cette interface existent, on veut écrire une classe `DisplayFormat` *qui permet de récupérer toutes les fonctionalités correspondant à tous les formats pour lesquels une telle classe existe.*

Ce programme doit être capable de récupérer toutes les classes dans un répertoire donné (disons, dans un premier temps, le répertoire courant) et de trier celles qui ont un certain nom (disons, qui commencent par `Format`*). Il devra de plus créer des instances de ces classes, puis appeler sur ces objets des méthodes définies dans leur classe respective.*

À terme, le programme sera capable d'utiliser, dynamiquement, les fonctionalités de tous les formats disponibles en recherchant toutes les classes de format et en appelant les méthodes définies sur des objets instances de ces classes (sans pour autant les connaître statiquement à la compilation).

1. *Écrire l'interface* `Format` *avec une méthode* `affiche()` *qui se contente d'afficher le format courant*[1], *et les classes* `FormatHTML`, `FormatPDF` *et* `FormatTEX` *implantant cette interface, avec les définitions de méthodes correspondantes.*
2. *Écrire la classe* `DisplayFormat`, *et une méthode* `formatFileNames()` *qui récupère dans le répertoire courant tous les noms de fichiers commençant par "Format" et ayant l'extension ".class" (définir pour cela une classe* `MyFormatFilter` *qui implante l'interface* `java.io.FilenameFilter`*). Cette méthode retourne un tableau de String.*
3. *Écrire une méthode* `formatClassNames()` *qui retourne un tableau de classes associées à chaque nom de classe de format.*
4. *Écrire une méthode* `afficheFormats` *qui, pour chaque classe de format disponible, créé une instance de la classe et appelle sur cet objet la méthode* `affiche()`.

[1] On peut imaginer des méthodes plus intéressantes permettant de faire des conversions ou de sauvegarder des fichiers dans un format considéré.