ÜNIVERSITÉ de Marne-la-Vallée

Travaux Dirigés de Java n°6

—Maîtrise d'informatique—

▸ **Exercice 1. (Qui suis-je ?)**

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

- *à une instance* `String s = "toto"`*;*
- *au nom de la classe* `String`*;*
- *à une instance* `Integer i = new Integer(2)`*;*
- *à un entier primitif* `int i = 2`*;*
- *à une instance* `Vector v = new Vector()`*;*
- *au nom de la classe* `Vector`*;*
- *au nom de la classe* `Disque` *définie au TD 3 ;*
- *au nom de l'interface* `Cloneable`*;*
- *au nom de l'interface* `Stack` *définie au TD 4.*

▸ **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` *ni utiliser explicitement la classe ni le type* `Date`*).*

- *À partir de cet objet classe, créez une nouvelle instance et affichez là. Comment expliquez vous le résultat?*
- *Récupérez le constructeur (*`java.lang.reflect.Constructor`*) à un argument de type* `long` *de cette classe. Créez une instance de la classe correspondant à la valeur d'argument 0 et affichez là.*
- *Quelle est la date maximale représentable dans ce format?*
- *Avec ce constructeur, créez deux instances* `d1` *et* `d2` *correspondant aux valeurs d'argument* `100000` *et* `100001`*. Pour tester si* `d1` *est "avant"* `d2` *au sens de ce format, récupérez par réflexion la méthode* `before()` *(object de la classe* `java.lang.reflect.Method`*) qui accepte un seul argument de type* `Date`*, et invoquez-la avec comme cible* `d1` *et comme argument* `d2`*.*

▸ **Exercice 3. (Problèmes d'exclusion mutuelle)**

Écrire une classe `Point` *avec deux champs entiers (*`x` *et* `y`*), une méthode* `void moveTo(int x, int y)` *(qui déplace le point aux coordonnées passées en argument) et une méthode* `toString()` *qui affiche ce point au format* `(x,y)`*.*

Écrire une classe `Init` *destinée à être exécutée dans un processus léger (*`Thread`*), c'est-à-dire implantant l'interface* `Runnable`*. Cette classe* `Init` *contient un constructeur à deux arguments* `Init(Point p, int val)`*. Sa méthode* `run()` *déplace* `p` *en* `(val,val)`*, puis affiche le point, et ce dans une boucle infinie.*

Créer deux processus légers ayant pour cible (code à exécuter) deux objets de la classe `Init` *construits avec le même point* `p` *et des valeurs différentes (par exemple,* `Init(p,1)` *pour l'un et* `Init(p,2)` *pour l'autre).*

Démarrer les deux processus légers. Quels sont les différents affichages possibles pour le point ? Que faire ?

▸ **Exercice 4. (Producteur et consommateur)**

On désire développer une petite application simulant les comportements concurrents d'un ensemble de producteurs et de consommateurs de messages. D'un côté, les producteurs produisent des messages qu'ils stockent dans un buffer commun ; de l'autre, les consommateurs récupèrent dans ce même buffer les messages (dans l'ordre où ils y ont été placés).

On doit respecter un certain nombre de contraintes et pouvoir paramétrer l'application :

- *un producteur ne peut produire un message que si le buffer n'est pas plein et, bien sûr, un consommateur ne peut lire un message que si le buffer n'est pas vide (le producteur ou consommateur dans l'impossibilité de produire ou de consommer est mis en attente) ;*
- *le débit de chaque producteur et de chaque consommateur est paramétrable à sa construction (il représente le temps d'attente entre deux productions ou deux consommations).*

Écrire les classes `Producteur` *et* `Consommateur` *permettant de mettre en œuvre ces spécifications. On pourra implanter le buffer avec un* `Vector` *et chaque message produit doit être aisément identifiable. Chaque producteur et consommateur sera exécuté par un processus léger (*`Thread`*) différent, en concurrence avec les autres.*