Travaux Dirigés de Java n°4

— `Etienne.Duris@univ-mlv.fr` *(Merci à* `Olivier.Dedieu@inria.fr`*)*—

▸ **Exercice 1. (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 2. (Producteur et consommateur)**

On désire développer une petite application simulant les comportements concurrents d'un producteur et d'un consommateur de messages. D'un côté, un producteur produit des messages qu'il stocke dans un buffer ; de l'autre, un consommateur associé à ce producteur consomme les messages stockés dans le buffer. On doit respecter un certain nombre de contraintes et pouvoir paramétrer l'application :

- *la taille du buffer est fixée à la construction du producteur ;*
- *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 ;*
- *les accès au buffer de messages doivent être synchronisés entre le producteur et le(s) consommateur(s) ;*
- *les* débits *du producteur et de chaque consommateur sont paramétrable (ils doivent* attendre *une durée donnée 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 pourra être la date courrante. Chaque producteur et consommateur sera exécuté par un processus léger (*`Thread`*) différent, en concurrence avec les autres.*

▸ **Exercice 3. (Le dîner des philosophes)**

Cinq philosophes sont assis autour d'une table ronde, où ils réfléchissent et où ils mangent. Chacun dispose d'un bol de riz. Une baguette est disposée entre chaque philosophe et, pour pouvoir manger, un philosophe doit prendre successivement les deux baguettes qui sont de chaque côté de lui.

L'exercice consiste à modéliser ce problème de sorte que chaque philosophe puisse manger et réfléchir sans qu'aucun d'eux ne meure de faim.

▸ **Exercice 4. (Un tri multi-thread)**

Soit `t` *un tableau à trier. Implanter l'algorithme de tri suivant :*

1. *Soient* `i` *et* `j` *(*`i<=j`*) les indices de l'intervalle de* `t` *à trier.*
2. *Si* `(j-i < 2)` *et* `(t[i] > t[j])`*, échanger* `t[i]` *et* `t[j]`*.*
3. *Sinon :*
   - *soit* `k = i + (j-i)/2`*;*
   - *appliquer l'algorithme sur* `[i,k]` *dans un nouveau processus léger (*`Thread`*) ;*
   - *appliquer l'algorithme sur* `[k+1,j]` *dans un nouveau processus léger ;*
   - *attendre la fin des deux processus légers ;*
   - *faire une fusion triée des tris effectués par les deux processus légers ;*
   - *informer le processus léger parent que le tri sur* `[i,j]` *est terminé.*