TP1 : threads

On rappelle qu'on désigne sous le nom de *processus* des programmes indépendants qui communiquent à l'aide du système d'exploitation. Dans ce cadre, on appelle *threads* les différents fils d'exécution qui peuvent exister au sein d'un même processus. Les threads (appartenant à un processus donné) peuvent communiquer plus directement que les processus, en effet, ils partagent leur espace d'adressage et leurs zones mémoires, à l'exception de leurs piles d'exécution (une par thread). Dans le cadre du langage Java le processus correspondant à un programme est l'instance de la machine virtuelle sur laquelle s'exécute le programme. L'exécution du programme est un thread (ainsi que, par exemple, le ramasse-miettes).

L'objectif de ce TP est de prendre en main les API Java permettant de gérer des threads (création, synchronisation, etc). Dans un premier temps on se consacrera à l'API de bas niveau (création et exécution de threads « à la main ») puis on verra qu'il existe une API de plus haut niveau, rendant plus agréable la manipulation des threads.

1 Préliminaires

Question. Récupérez sur Madoc le fichier **Pool.java**.

Ce fichier définit la classe *Pool* dont chaque instance permet de visualiser (il vous faudra quand même un peu d'imagination) le déplacement d'une balle (sans frottements) sur une table de billard.

Question. Créez un fichier **Appli.java** contenant une méthode *main* qui crée un objet de type *Pool* et appelle la méthode *go* de cet objet.

2 API de bas niveau

2.1 Interface Runnable

La première étape pour créer un thread consiste à créer un objet d'une classe implémentant l'interface *Runnable* (exécutable). Cette interface est assez simple puisqu'elle ne demande d'implanter qu'une seule méthode, publique : *run* (de type *void* et sans paramètres), qui décrit ce à quoi doit correspondre l'exécution d'un objet exécutable.

Question. Modifiez le fichier **Pool.java** pour faire en sorte que la classe *Pool* implémente *Runnable*. Une exécution d'un objet de cet classe devra permettre de visualiser la balle se déplaçant sur le billard (i.e. elle devra correspondre à un appel à la méthode *go*).

2.2 Classe Thread

À partir d'un objet d'une classe implémentant *Runnable* on peut ensuite créer un thread. Pour cela on crée un nouvel objet de la classe Thread en lui donnant en argument l'objet de la classe *Runnable* préalablement créé. On démarre ensuite l'exécution du thread en utilisant la méthode *start* de la classe Thread.

Question. Modifiez le fichier **Appli.java** préalablement créé pour que votre objet de type *Pool* soit exécuté comme un nouveau thread.

Question. Écrivez une classe *Controller* qui, étant donné un objet de type *Pool*, modifie de temps en temps la vitesse de la balle (ceci pourra se faire par un appel à la méthode *setSpeed* de la classe *Pool*). La classe *Controller* devra implémenter *Runnable*.

Question. Modifiez **Appli.java** pour créer un nouveau thread à partir d'un objet de la classe *Controller* (permettant de contrôler l'objet de type *Pool* correspondant à l'autre thread).

Question. Constatez que le comportement du premier thread est bien contrôlé à partir du second.

2.3 Synchronisation

Comme indiqué plus haut, les différents threads crées par un programme partagent leur mémoire. Il faut donc mettre en œuvre des mécanismes de protection pour éviter, par exemple, que deux threads écrivent en même temps au même endroit. Pour cela, on peut ajouter le mot clé *synchronized* aux méthodes utilisées pour modifier ou lire les variables partagées entre plusieurs threads.

Question. Modifiez la méthode *setSpeed* de la classe *Pool* en lui ajoutant le mot clé *synchronized*. (Et pour mieux vous rendre compte tout-à-l'heure de l'effet de ce mot clé, ajoutez un temps d'attente dans la méthode.)

Question. Ajoutez à votre méthode *main* la création d'un troisième thread à partir d'un objet de type *Controller* contrôlant lui aussi votre objet de type *Pool* (celui-ci doit donc maintenant être contrôlé par deux threads différents).

Question. Qu'en déduisez vous sur l'effet du mot clé *synchronized* ? Pour répondre à cette question, n'hésitez pas à faire afficher des informations avant et après l'appel à la méthode *setSpeed*, par exemple pour savoir qui essaye d'appeler cette méthode à quel moment.

3 API de haut niveau

Question. Consultez la documentation du paquet **java.util.concurrent**, intéressez vous notamment aux interfaces *Callable*, *Executor*, *ExecutorService* et *Future*. Quelles différences faites vous avec l'API de bas niveau ? Pouvez-vous refaire le début du TP en utilisant cette nouvelle API ?