Séance 8 – Threads

Objectifs

- Ecrire des threads avec la classe Thread et l'interface Runnable
- Se rendre compte des problèmes de deadlock
- Résoudre les problèmes de concurrence à l'aide du "synchronized"
- Se rendre compte des bénéfices du multithreading

Thèmes abordés

- Threads
- Concurrence

1. Compteur

Un compteur a un nom et permet de compter de 1 à n (nombre entier positif fourni lors de la création du compteur). Entre chaque nombre, le compteur marque une pause aléatoire (de 0 à 5000 millisecondes par exemple). Il affiche alors son nom et le nombre (par exemple, "Bolt : 7"). À la fin, il affiche un message du type "Bolt a fini de compter jusqu'à xx". Les classes à compléter se trouvent dans le package `compteur`.

a) Classe Thread

La première méthode pour implémenter un thread est d'étendre la classe prédéfinie Thread :

1. Complétez la méthode `run()` de la classe `CompteurThread` qui hérite de Thread. Le nom du compteur qui a gagné doit être affiché (attendre que tous aient fini). Le compteur gagnant doit en plus afficher afficher un message du type "Bolt a gagné". Il faut conserver pour cela l'instance gagnante dans un attribut de classe `CompteurThread`. Mais attention cet attribut peut être lu/modifié par plusieurs threads en même temps.
2. Complétez la classe de tests `TestCompteurThread` qui lance plusieurs compteurs qui comptent jusqu'à 10. Voyez celui qui a fini le plus vite.

b) Interface Runnable

L'héritage à partir de Thread est contraignant car il empêche tout autre héritage. Une deuxième manière de faire est d'implémenter l'interface Runnable. Ceci permet l'héritage d'une autre classe et l'implémentation d'autres interfaces.

1. Complétez la classe `CompteurRunnable` qui étend l'interface Runnable. Cette fois-ci, on ne cherche plus le gagnant. À la place, chaque compteur va devoir afficher son ordre d'arrivée : le message de fin est du type : "Bolt a fini de compter en position 3". La position pourrait être conservée dans un attribut de classe de type `int`. Mais attention cet attribut peut être lu/modifié par plusieurs threads en même temps. Des types spéciaux, dits atomic, peuvent être utilisés dans ce genre de cas. Pour plus d'information, rendez-vous sur la documentation du package java.util.concurrent.atomic.
2. Utilisez la classe `TestCompteurRunnable` qui lance plusieurs compteurs qui comptent jusqu'à 10 pour voir le résultat.

2. Peer2Peer – deadlock

Un réseau en Peer to Peer (P2P) permet à chaque machine connectée d'être à la fois server et client. Cela permet à deux ordinateurs de se connecter directement sans l'intermédiaire d'un server. La classe `Peer` du package `peer2peer` tente de simuler le comportement de la connexion d'un peer à un autre sur un tel réseau.

La méthode `connecter(Peer peer)` permet de se connecter à un autre `Peer`, et de le garder en mémoire dans un set qui retient toutes les connexions du `Peer`. Ensuite, le `Peer` en paramètre essaie à son tour de se connecter au `Peer` qui vient de se connecter. Testez cette classe avec `TestPeer`. Vous remarquerez que toutes les connexions ne se font pas, et que le programme « bloque » : c'est un deadlock.

Essayez de comprendre d'où vient le deadlock et modifiez la méthode `connecter(Peer peer)` pour corriger le problème.

3. Abonnements – race condition

Observons maintenant la problématique de la race condition.

Dans le package `abonnements`, vous trouverez 4 classes :

a. `Compte` : cette classe contient un solde et un historique des dépense. Sa méthode `depenser` permet d'enregistrer une dépense et de modifier son solde. Sa méthode `verifier` permet d'afficher l'historique pour vérifier que les opérations sont bien légales.
b. `Depense` : cette classe représente une dépense. Elle a un nom et un montant.
c. `Abonnement` : un abonnement est une dépense régulière (ici mensuelle) à enregistrer sur un compte. Les abonnements sont des threads qui s'exécutent tant que le solde du compte est suffisant.
d. `TestAbonnements` : cette classe crée 5 abonnements pour un compte, lance les threads, attend la fin de leur exécution et puis vérifie l'état du compte.

Problè-me

Le solde du compte ne peut pas être négatif puisqu'un abonnement s'arrête si le solde n'est pas suffisant pour le payer. Malheureusement, on peut voir que ce n'est pas le cas ici : des abonnements réussissent à prendre de l'argent malgré tout et le solde final est bien souvent négatif !

Modifiez le programme pour éliminer ce problème de race condition.

4. Tirage de cacahuètes

Le temps des fêtes de fin d'années arrive bientôt, et l'école souhaite organiser un tirage de cacahuètes pour apporter un peu de joie aux étudiants avant le blocus. La direction a donc demandé à des étudiants motivés (vous) de préparer un programme qui va tirer les cacahuètes.

Un tirage de cacahuètes se fait comme suit : les participants sont enregistrés par groupes dans le programme. Le programme pioche ensuite les participants un par un, et leur « tire une cacahuète », c'est-à-dire qu'il leur attribue un autre participant à qui il faudra offrir un petit cadeau. Afin de favoriser les rencontres, un participant n'aura jamais une cacahuète au sein du même groupe que lui.

Concrètement, on vous fournit dans le package tirage une série de classes :

- `Participant` : représente un participant. Déjà implémenté.
- `Groupe` : représente un groupe de participants. Déjà implémenté.

- `Tirage` : la classe principale, qui contient tous les participants et tous les groupes, et qui sera responsable d'effectuer le tirage à proprement parler. `TestTirage` : une classe de tests

1. Commencez par compléter les méthodes `ajouterGroupe` et `ajouter` de la classe `Tirage`. Vous pouvez tester cette partie avec la classe de tests.

2. Créez ensuite une classe interne à `Tirage` qui étend Thread. Dans cette classe, vous effectuerez le tirage. Étant donné qu'il y a de nombreux étudiants dans la haute école, il est normal d'avoir besoin de threads pour améliorer l'efficacité du programme.
Dans leurs méthodes `run()`, les threads de tirage appelleront une méthode `tirer` sur chaque participant qui n'a pas encore de cacahuète. Il est recommandé d'utiliser l'attribut `participantsSansCacahuete` de la classe `Tirage` pour sélectionner le prochain participant qui doit tirer.
Voici les spécifications de la méthode tirer :

```
/**
 * Effectue le tirage d'un participant parmi ceux qui n'ont pas encore été tirés.
 * Il faut veiller à ne pas tirer de participants des groupes du @participant.
 * Il faut mettre la paire (@participant, participant tiré) dans la map des résultats.
 *
 * @param participant un participant à qui il faut attribuer une cacahuète.
 * @throws TirageImpossibleException si le tirage est impossible
 * Conseil : utilisez l'attribut membresTires de TriageImpl
 */
private void tirer(Participant participant) throws TirageImpossibleException
```

Remarquez qu'elle lance une `TirageImpossibleException` s'il est impossible de tirer une cacahuète pour le participant en paramètre. Dans ce cas, le thread doit simplement s'arrêter : tout le monde ne pourra pas offrir de cadeau, tant pis (ça leur fera des économies).

Cette méthode est difficile à implémenter. Suivez les étapes suivantes pour y parvenir :

1. Définissez l'ensemble des groupes auxquels le participant en paramètre appartient. Ce participant ne peut pas tirer de cacahuète appartenant à l'un de ces groupes.
2. Filtrez l'ensemble des participants pour obtenir une liste de participants qui ne contient que des participants qui :
    a. n'appartiennent pas à l'ensemble des groupes trouvés en 1,
    b. ne sont pas le participant en paramètre,
    c. qui n'ont pas encore été tirés (utilisez l'ensemble `participantsTires`).
3. Si le résultat est une liste vide, alors l'exception doit être lancée.
4. Sinon, vous pouvez sélectionner un participant au hasard dans la liste trouvée, l'ajouter dans l'ensemble des participants tirés, et ajouter la paire (participant, participant tiré) dans la map des résultats du tirage.

Attention à la synchronisation !

Pour tester votre tirage, vous pouvez décommenter la deuxième partie de la classe de tests. Cette partie créera 3 threads qui effectueront le tirage.