Développement informatique avancé orienté application - TP

Programmation concurrente

Virginie Van den Schrieck Arnaud Dewulf

La théorie relative à la programmation concurrente peut être trouvée sur le site Inginious, section 3.5.

1 Création de threads

Créez une classe `MyThread` qui étend la classe `Thread`. Avec cette classe, nous allons examiner les différentes états des threads. Pour cela, nous créerons deux threads qui s'exécuteront en parallèle et imprimeront leurs états respectifs à différents moments grâce à la méthode `getState()` de la classe `Thread`.

La classe `MyThread` possède les éléments suivants :

- Un constructeur avec un nom en paramètre, qui se contente d'appeler le constructeur de la classe mère correspondant
- Un attribut `Thread` appelé `other`
- Une méthode `setOther(Thread o)`
- Une méthode `run()`, qui va imprimer dix fois à la console les deux lignes suivantes :
  - Une première ligne avec le nom du thread courant et son état
  - Une seconde ligne avec le nom du thread courant, suivi du nom du thread `other` et de l'état de `other`

  Le but est qu'à chaque exécution du thread courant, il imprime son état ainsi que l'état de l'autre thread.

Créez ensuite une méthode `main()`, qui va effectuer les actions suivantes :

- Créer deux objets `MyThread`, avec chacun un nom différent (ex : threadA et threadB)
- Attribuer à chacun de ces deux objets l'autre thread comme attribut `other` grâce au setter correspondant
- Imprimer les états de ces deux threads à ce moment précis
- Démarrer les deux threads
- Imprimer les états de ces deux threads après leur démarrage

Exécutez plusieurs fois votre programme, et observez les résultats. Par quels états passent vos threads ? Pourquoi y-a-t'il chaque fois des résultats différents ?

2 Gestion des accès concurrents

1. Ecrivez une classe `BankAccount`, qui possède un solde comme attribut et trois méthodes : `deposit(double amount)`,`withdraw(double amount)` et `double getAmount()`. Il n'y a pas de contrainte sur le montant à retirer, le solde peut être négatif.
2. Créez ensuite deux classes étendant `Thread` : `Salary` et `BankCard`. Chacune possède un objet `BankAccount` comme variable d'instance. La méthode `run` de la première effectue une boucle de 100 itérations, chaque itération effectuant un versement de 10 euros sur le compte en banque. La méthode `run` de la première effectue elle une boucle de 100 retraits de 10 euros.
3. Créez une classe appelée AccountOwner, possédant une méthode `main`. Dans cette méthode `main`, créez un compte en banque, puis un objet Salary et un autre objet BankCard, en leur liant l'ojet `BankAccount` créé juste avant.
4. D'après vous, si vous lancez les deux threads, puis lisez le solde du compte banquaire une fois que ces deux threads sont terminés, quel solde devriez-vous obtenir ?
5. Lancez les deux threads `Salary` et `BankAccount`. Vérifiez le solde obtenu après exécution de ces threads.
6. Reproduisez 100 fois cette expérience, et affichez la moyenne des soldes obtenus. Qu'en déduisez-vous ?
7. Modifiez la classe `BankAccount` pour supprimer ces incohérences.

3 Contrôler une communication réseau avec des threads

Reprenez le programme de chat créé lors du TP sur les sockets. Dans cette version, chaque interlocuteur devait attendre son tour avant de parler. Modifiez le programme de sorte que l'envoi de message soit possible à tout moment.