TP noté n° 2 : Groupe INFO 2

13 décembre

Durée : 1h30 heures

Important : Ce TP est à rendre sur Moodle avant l'heure indiquée au tableau, le dépôt sera ensuite clos. Merci de rendre une archive tar ou zip (pas de rar) contenant uniquement du code java, avec un ou des `main` pour pouvoir effectuer les tests. Rédigez toutes les explications utiles à la compréhension de vos choix dans des commentaires dans le code.

Exercice 1 :

1. Écrivez une classe `Compteur` encapsulant un entier et disposant des méthodes : `incremente()`, `decremente()`, `getValeur()` qui permettent de modifier l'entier caché ou d'accéder à sa valeur.
2. Écrivez dans une classe `Exo1`, une méthode `static int calcule(int n, int t)`. Son rôle est de créer `t` threads identifiés par un `id` invariant différent dans $[0..n[$. Chaque thread modifiera un unique `Compteur` (le même partagé entre tous les thread) de la façon suivante :
   - chaque thread pairs (id=0,2,4,...) incrémente le compteur `n` fois.
   - chaque thread impairs (id=1,3,5,...) décrémente le compteur `n` fois.

   Vous illustrerez 2 méthodes différentes de création de thread
   La méthode retourne la valeur contenu dans le compteur lorsque tous les threads sont terminés. (Pour indication, cette valeur devrait être 0 si le nombre `t` est pair, ou `n` sinon. Utilisez la synchronisation appropriée pour obtenir le comportement demandé, en expliquant tout naturellement
3. Donnez un exemple d'utilisation de cette méthode dans un main pour
   - `n` = 1000000 et `t` = 4
   - `n` = 1000000 et `t` = 5

Exercice 2 :

1. Écrivez une classe `MyOptional<E>` encapsulant un objet de type `E` (pouvant être temporairement `null`) et disposant des méthodes suivantes :
   - `boolean isPresent()` qui retourne `true` ssi une valeur est présente ;
   - `E getValeur()` qui renvoie la valeur supposée présente (pas la peine de le vérifier).
   - `void setValeur(E valeur)` qui permet de modifier la valeur encapsulée.
   - un constructeur sans argument
2. Dans une méthode statique `void testMyOptional(int n);`
   - créez `n` threads personnalisés qui possèdent deux attributs `entree` et `sortie` de type `MyOptional<Integer>`.

– créez `n-1` `MyOptional<Integer>` optional1 et optional2 ... tels que la `sortie` de premier thread et l'`entree` du deuxième thread partagent l'objet optional1, de même la `sortie` du deuxième thread et l'`entree` de troisième thread partagent optional2, etc ... comme on peut le voir sur cette figure :

– chaque thread se comporte de la même façon : il attend que l'optional de son `entree` ait une valeur, une fois qu'elle est disponible il la récupére, la multiplie par 2, attend quelques instants et place ce résultat dans l'optional de sa `sortie`.
Si le thread n'a pas d'`entree` définie (comme c'est le cas pour le premier thread de la figure), il n'a rien à attendre et décide de placer la valeur 2 dans l'optional de sa `sortie`.
Si le thread n'a pas de `sortie` définie (comme on peut le voir pour le troisième thread), puisqu'il n'a pas de message à transmettre il se contentera d'afficher la valeur calculée et aura terminé.

3. Donnez un exemple d'utilisation.