Java Avancé
TD – Le problème des lecteurs-rédacteurs

Nawfal Massine MALKI, 4A STI, TD2

Introduction

Ce TD traite de l'utilisation d'une ressource partagée de la mémoire entre plusieurs threads. Nous distinguons ici deux actions possible : lire la ressource ou écrire dans la ressource.

La lecture s'effectue par une instance de la classe Reader.

L'écriture s'effectue par une instance de la classe Writer.

Un rédacteur ne peut pas écrire quand un lecteur lit la ressource et un lecteur ne peut pas lire quand un rédacteur écrit dans la ressource.

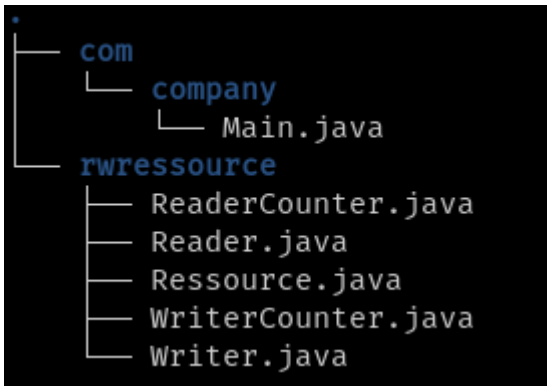
Plusieurs lecteurs peuvent lire en même temps, en revanche un seul et unique rédacteur peut modifier la ressource.

Deux possibilités : protéger l'accès mémoire à la ressource ou alors protéger la méthode qui permet de lire ou d'écrire dans la ressource.

Nous avons opté pour le deuxième choix dans ce TD car il permet plus de flexibilité dans la distinction des actions de lecture et d'écriture.

Implémentation générale (en bref)

Nous disposons des classes suivantes :



Ressource : contient la ressource en question ainsi que les méthodes read() et write() qui permettent d'accéder à ou de modifier la ressource. C'est ici que l'on va implémenter la logique du problème. En bref :

- On utilise des locks ReentrantLock pour protéger l'accès simultané aux méthodes read() et write() par les threads.

- On utilise une sémaphore pour limiter le nombre de lecteurs simultanés.

- On utilise le modèle **push** pour la gestion des signaux avec la classe Condition de ReentrantLock : quand un service est occupé, il laisse le thread attendre que celui-ci soit libéré avec les méthodes condition.await() pour mettre en attendre et condition.signalAll() pour signaler la libération du service.

Reader : classe du thread lecteur, sa méthode read() lui permet de lire la ressource, si la ressource le lui permet.

Writer : classe du thread rédacteur, sa méthode write() lui permet d'écrire dans la ressource, si la ressource le lui permet.

ReaderCounter : classe du compteur de lecteurs, compte le nombre de lecteurs qui accèdent à la ressource simultanément.

- Ses méthodes increment() et decrement() permettent d'agir sur le compteur.

- Sa méthode increment() interdit l'accès à la ressource si on dépasse le nombre de lecteurs autorisés.

WriterCounter : classe du compteur de rédacteurs, compte le nombre de rédacteurs (1 max à chaque fois). S'il y a un rédacteur qui modifie la ressource, le compteur passe à 1 et la classe Ressource refuse de donner l'accès à un lecteur ou à un rédacteur.

1.a. Priorité aux rédacteurs

Ici, un rédacteur a la priorité sur les lecteurs. On formalise cette condition comme ce qui suit.

La méthode Ressource.write()

Quand un rédacteur appelle la méthode **write()** :

- il incrémente le writerCounter

- mobilise la ressource

- la modifie

- décrémente le writerCounter

- libère la ressource

- signale aux autres threads rédacteurs que la ressource est libre (Condition **freeToWrite**.signallAll())

```
try {
    this.writerCounter.increment();
    lock.lock();
    this.myRessource = newRessource;
    this.writerCounter.decrement();
    if (this.writerCounter.getNbWriters() == 0) {
        this.freeToWrite.signalAll();
    }
} finally {
    lock.unlock();
    return true;
}
```

La méthode Ressource.read()

La méthode read() vérifie le nombre de rédacteurs en cours grâce à la méthode getNbWriters() de la classe WriterCounter. S'il y a au moins un rédacteur, on fait attendre le thread lecteur.

```
// We make sure that no writer is currently writing
lock.lock();
while(this.writerCounter.getNbWriters() > 0) {
    System.out.println("Reader is trying to read, but a writer is actually writing");
    this.freeToWrite.await();
}
lock.unlock();
```

Output : (plusieurs lecteurs et rédacteurs)

```
int nbReaders = 5;
int nbWriters = 3;
int maxReadersAllowed = nbReaders;
```

```
Reader is trying to read, but a writer is actually writing
Writer 7 writes: Last modified by Writer 7
Reader 1 reads: Last modified by Writer 1
Writer 1 writes: Last modified by Writer 1
Writer 10 writes: Last modified by Writer 10
Writer 8 writes: Last modified by Writer 8
Reader 2 reads: Last modified by Writer 8
Writer 3 writes: Last modified by Writer 3
```

1.b. Nombre borné de lecteurs

La classe ReaderCounter va nous permettre de compter le nombre de lecteurs simultanés et de les limiter.

```
18      public void increment() {
19          lock.lock();
20
21          // If maxReaders is reached, a reader thread waits for the signal allowing him to read
22          while (nbReaders >= maxReadersAllowed) {
23              try {
24                  System.out.println("Maximum Readers allowed (" + maxReadersAllowed + ") reached :
    " + nbReaders);
25                  this.maxReadersAllowedReached.await();
26              } catch (InterruptedException e) {
27                  e.printStackTrace();
28              }
29          }
30          this.nbReaders++;
31          lock.unlock();
32      }
33
34      public void decrement() {
35          lock.lock();
36          this.nbReaders--;
37          if (nbReaders < maxReadersAllowed)
38              this.maxReadersAllowedReached.signal();
39          lock.unlock();
40      }
41  }
```

Dans la méthode Ressource.read() on utilise une sémaphore que l’on initialise au nombre max de lecteurs. Chaque lecteur, quand il appelle la méthode read(), récupère un jeton pour pouvoir accéder au contenu de la variable puis le relâche après lecture.

```
72              // We make sure that maxReadersAllowed isn't reached
73              s.acquire();
74              this.readerCounter.increment();
75              return this.myRessource;
76          } finally {
77              s.release();
78              this.readerCounter.decrement();
79          }
```

Output :

la solution est fonctionnelle mais j’arrive rarement à avoir plus d’un lecteur simultané. Je n’ai pas pris de screenshot au bon moment.

2. Un rédacteur par ressource

Pour permettre à une ressource d'être modifiée que par un seul et unique rédacteur, nous avons décidé d'implémenter ça dans la classe Ressource. Celle-ci aura un attribut hasOwner et un attribut ownerName. Si hasOwner = true alors seul ownerName a le droit de modifier la ressource.

```
44      public boolean write(String newRessource, String writerName) throws InterruptedException {
45          if (hasUniqueOwner && !ownerName.equals(writerName)) {
46              System.out.println(writerName + " is not allowed to write in the ressource. Aborting..
    .");
47              return false;
48          }
```

Output :

```
int nbReaders = 5;
int nbWriters = 3;
int maxReadersAllowed = nbReaders;
WriterCounter writerCounter = new WriterCounter();
ReaderCounter readerCounter = new ReaderCounter(maxReadersAllowed);
Ressource r = new Ressource(maxReadersAllowed, readerCounter, writerCounter, ownerName: "Writer 1");
```

```
Reader 3 reads: Blank for now
Reader 2 reads: Blank for now
Writer 1 writes: Last modified by Writer 1
Reader 1 reads: Blank for now
Reader 4 reads: Blank for now
Writer 3 is not allowed to write in the ressource. Aborting...
Writer 2 is not allowed to write in the ressource. Aborting...
Reader 5 reads: Blank for now
Reader 3 reads: Last modified by Writer 1
Reader 2 reads: Last modified by Writer 1
```