

# Readme

---

Auteurs: Thomann Yanick et Gallay David

Date: 30 mars 2021

## Issues

---

Les asserts sont faits de manières synchrone, mais les threads interviennent de manière asynchrone. Nous ne sommes pas non plus à l'abri des faux-positifs, aussi les tests ont été lancés manuellement plusieurs fois

## 1ère solution

```
1 redacteur2.stopWrite();
2
3 // après les redacteurs , lecteur3 est libéré
4 assertFalse(lecteur3.isWaiting());
```

Dans cette exemple, le stopWrite déclenche la mise en route de lecteur3, mais il est possible que lecteur 3 ne soit pas encore arrivé suffisamment loin à ce moment. Pour fixer le problème temporairement, un délai supplémentaire est accordé après le notify:

```
1 private synchronized void notifyLecteursRedacteurs() {
2     synchronized (writeLock) {
3         writeLock.notifyAll();
4     }
5     sleep();
6     synchronized (readLock) {
7         readLock.notifyAll();
8     }
9     sleep();
10 }
11
12 private void sleep() {
13     try {
14         Thread.sleep(100);
15     } catch (Exception e) {
16         System.out.println(e);
17     }
18 }
```

# Solution Finale

## Utilisation de boucles d'attentes

```
1 // Si le Lecteur obtient le droit d'écriture, il passera la valeur reading
  à true
2 // Sinon, le thread finira par passer en attente
3 while(reading == false && thread.getState() == Thread.State.RUNNABLE);
4
5 /* La même boucle est utilisée après le start ainsi qu'en début le
  isWaiting.
6 * Les 2 valeurs doivent être cohérentes, malheureusement, on ne peut pas
  compter que sur l'un de ces états:
7 * Il peut arriver que la boucle soit trop longue à définir la variable
  reading
8 * ou que l'état du thread ne soit pas encore définit
9 */
10
11 /* Un compteur de redacteur en attente est utilisé pour savoir si un
  rédacteur attend.
12 * Si c'est le cas, alors nous ne débloquons que les rédacteurs et
  attendons que l'un d'eux se connecte.
13 * Sinon, nous débloquons les lecteurs
14 */
15
16 private synchronized void notifyLecteursRedacteurs() {
17
18     synchronized (waitingRedacteurCount) {
19         if(waitingRedacteurCount > 0) {
20             synchronized (writeLock) {
21                 writeLock.notifyAll();
22             }
23             while(redacteur == null);
24         }
25     }
26     synchronized (readLock) {
27         readLock.notifyAll();
28     }
29 }
30
```

Il y a évidemment un risque de ce retrouver avec un boucle infini, si par exemple le thread vient à être interrompu, les valeurs reading/writing ne repasseraient jamais à false. C'est une des raisons pour lesquels

1. Lecteurs/Redacteurs n'héritent pas de Thread mais en possède une instance d'un sous-classe anonyme.
2. Nous utilisons le State RUNNABLE et nous WAITING pour nos vérifications.

Edit: 30 tests sur 30 réussis au travers d'Idea. Au moment de faire valider par travis, nous observons que le problème n'était pas résolu avec la commande `mvn clean test`. Nous avons donc remis un sleep après les notifyall. Le test est à présent valider à chaque fois mais cette solution est très circonstancielle et n'apporte aucune garantie dans un programme plus complexe.

# Links

---

- [Travis CI](#)
- [Debug.Idea](#)