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'abris des faux-positifs, aussi les tests ont été lancés manuellement plusieurs fois

1ère solution

```
redacteur2.stopWrite();

// après les redacteurs , lecteur3 est libéré
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 suffisament loin à ce moment. Pour fixer le problème temporairement, un délai supplémentaire est accordé après le notifiy:

```
private synchronized void notifyLecteursRedacteurs() {
    synchronized (writeLock) {
        writeLock.notifyAll();
    }
    sleep();
    synchronized (readLock) {
        readLock.notifyAll();
    }
    sleep();
}

private void sleep() {
    try {
        Thread.sleep(100);
    } catch (Exception e) {
        System.out.println(e);
    }
}
```

Solution Finale

Utilisation de boucles d'attentes

```
// Si le Lecteur obtient le droit d'écriture, il passera la valeur reading à true
// Sinon, le thread finira par passer en attente
while(reading == false && thread.getState() == Thread.State.RUNNABLE);

/* La même boucle est utilisée après le start ainsi qu'en début le isWaiting.
 * Les 2 valeurs doivent être cohérentes, malheureusement, on ne peut pas compter que sur l'un de ces états:
 * Il peut arriver que la boucle soit trop longue à définir la variable reading
 * ou que l'état du thread ne soit pas encore définit
 */

/* Un compteur de redacteur en attente est utilisé pour savoir si un rédacteur attent.
 * Si c'est le cas, alors nous ne débloquons que les rédacteurs et attendons que l'un d'eux se connecte.
 * Sinon, nous débloquons les lecteurs
 */

    private synchronized void notifyLecteursRedacteurs() {

        synchronized (waitingRedacteurCount) {
            if(waitingRedacteurCount > 0) {
                synchronized (writeLock) {
                    writeLock.notifyAll();
                }
                while(redacteur == null);
            }
        }
        synchronized (readLock) {
            readLock.notifyAll();
        }
    }

```

Il y a évidement un risque de ce retrouver avec un boucle infini, si par exemple le thread vient à être interrompu, les valeurs reading/writing ne repasserait 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.

Links

- Travis CI
- Debug Idea