Exercice de programmation concurrente: Verrou lecteurs-rédacteurs réentrant

Donner l'implémentation d'une classe Java nommée `MyReentrantReaderWriterLock` permettant de synchroniser les accès à une ressource partagée. La sémantique de synchronisation souhaitée est de type *lecteurs-rédacteurs* : plusieurs lecteurs peuvent accéder de manière concurrente à la ressource mais un rédacteur doit disposer d'un accès exclusif.
La classe doit implémenter l'interface suivante :

```
/** Demande d'acquisition du verrou (potentiellement bloquante) en mode lecteur */
void readerLock() throws InterruptedException;

/** Libération du verrou obtenu en mode lecteur */
void readerUnlock();

/** Demande d'acquisition du verrou (potentiellement bloquante) en mode rédacteur */
void writerLock() throws InterruptedException;

/** Libération du verrou obtenu en mode rédacteur */
void writerUnlock();
```

Les spécifications précises demandées sont les suivantes :

- La priorité est donnée aux rédacteurs : aucun nouveau lecteur ne doit obtenir l'accès à la ressource lorsqu'il y a au moins un rédacteur qui attend de pouvoir y entrer.
- Le verrou doit être réentrant : un thread qui détient le verrou (en mode lecteur ou rédacteur) peut redemander et obtenir le verrou (dans le même mode) sans être bloqué.
- Un thread qui détient le verrou en mode rédacteur peut demander et obtenir sans bloquer le verrou en mode lecteur.
- La séquence d'appels suivante (pour un même thread) est autorisée : `writerLock(); readerLock(); writerUnlock(); readerUnlock();`
- Un thread qui détient le verrou en mode lecteur n'est pas autorisé à faire directement une demande du verrou en mode rédacteur ; il doit d'abord libérer le verrou en mode lecteur.

On suppose par ailleurs que les utilisateurs de la classe `MyReentrantReaderWriterLock` en font un usage correct, et respectent donc les règles ci-dessus ainsi que les suivantes :

- un thread qui appelle la méthode `readerLock` effectue toujours un appel ultérieur à la méthode `readerUnlock` pour libérer le verrou ;
- un thread qui appelle la méthode `readerUnlock` a toujours préalablement effectué un appel à la méthode `readerLock` pour obtenir le verrou ;
- mêmes principes pour `writerLock` et `writerUnlock`.

Il n'est donc pas demandé de détecter et prendre en charge les cas d'utilisations incorrectes dans le code de la classe.

On rappelle l'existence de méthodes suivantes de la classe `Thread` :
— `static Thread currentThread()` : retourne un référence vers l'objet `Thread` associé au thread appelant
— `long getId()` : retourne l'identifiant unique ("thread ID") associé à un thread

Remarques :
— Une solution qui peut engendrer des réveils inutiles de threads est acceptable dès lors que cela est clairement indiqué (via un simple commentaire dans le code).
— Une solution qui ne prend pas en compte le problème des réveils intempestifs (*spurious wakeups*) de threads est acceptable dès lors que cela est clairement indiqué (via un simple commentaire dans le code).