TP5 : Volatile et opérations atomiques

Exercice 1 - A vos chronometres

1. Ce programme créer et lance une thread qui va boucler tant que stop est à false. Le programme attend 100 millisecondes puis positione stop à true. Enfin il arrête la thread. Cepdendant il n'y a pas de synchronisation, la boucle peut alors durer jusqu'à l'infini.

2. On peut modifier rajouter un block synchronized.

```
public void runCounter() {
    var localCounter = 0;
    for(;;) {
        synchronized (lock) {
            if (stop) {
                break;
            }
            localCounter++;
        }
    }
    System.out.println(localCounter);
}
```

3. Les implantations qui n'ont ni blocs syncrhonized ni lock sont appelé lock-free.

```
private volatile boolean stop;
```

Exercice 2 - Exercice 2 - SpinLock

1. **réentrant** veut dire qu'un thread peux utiliser un jeton qu'il a déjà utilisé.

2. Ce code créer et lance 2 **thread** qui vont appelé un runnable qui incrémente un compteur 1_000_000 de fois pour chaque **thread** puis affiche à la fin le résultat du compteur. Si on a 2 **thread** comme ici, le compteur vaudra donc 2_000_000 si la classe est **thread-safe**. La classe ici n'est pas **thread-safe** car lors de la lecture et de l'écriture du compteur, une autre **thread** peut écrire dans le compteur. **volatile** ne permet pas de rendre les opérations atomiques.

3. Si on arrive pas à acquérir le lock, il faut attendre. Le problème est que l'on fait de l'attente active. **Thread.onSpinWait** permet de faire de l'attente passive.

4.
```
private volatile boolean token;
private final static VarHandle varHandle;
static {
 Lookup lookup = MethodHandles.lookup();
 try {
     varHandle = lookup.findVarHandle(SpinLock.class, "token", boolean.class);
 } catch (NoSuchFieldException | IllegalAccessException e) {
     throw new AssertionError(e);
```

```
 }
}

public void lock() {
 for (;;) {
     if (varHandle.compareAndSet(this, false, true)) {
         return;
     }
     Thread.onSpinWait();
 }
}

public void unlock() {
 token = false;
}
```

Exercice 3 - Generateur pseudo-aléatoire lock-free

1. Un **générateur pseudo-aléatoire** fonctionne avec une **graine** prédéfinis.
   L'implantation donnée n'est pas **thread-safe** car deux appels à la méthode **next** peut mener à un état incohérent de la mémoire.

2. ```
   private final AtomicLong x;

   public RandomNumberGenerator(long seed) {
       if (seed == 0) {
           throw new IllegalArgumentException("seed == 0");
       }
       x = new AtomicLong(seed);
   }

   public long next() {  // Marsaglia's XorShift
       for (;;) {
           var x = this.x.get();
           var newX = x;
           newX ^= newX >>> 12;
           newX ^= newX << 25;
           newX ^= newX >>> 27;
           if (this.x.compareAndSet(x, newX)) {
               return newX * 2685821657736338717L;
           }
       }
   }
   ```

3. ```
   public static long nextValue(long previous) {
       previous ^= previous >>> 12;
       previous ^= previous << 25;
       previous ^= previous >>> 27;
       return previous;
   }
   public long next() {  // Marsaglia's XorShift
   ```

```
    return this.x.updateAndGet(RandomNumberGenerator::nextValue) *
2685821657736338717L;
 }
```

4.
```
 private volatile long x;
 private final static VarHandle varHandle;

 static {
     Lookup lookup = MethodHandles.lookup();
     try {
         varHandle = lookup.findVarHandle(RandomNumberGenerator.class, "x",
long.class);
     } catch (NoSuchFieldException | IllegalAccessException e) {
         throw new AssertionError(e);
     }
 }

 public RandomNumberGenerator(long seed) {
     if (seed == 0) {
         throw new IllegalArgumentException("seed == 0");
     }
     x = seed;
 }

 public static long nextValue(long previous) {
     previous ^= previous >>> 12;
     previous ^= previous << 25;
     previous ^= previous >>> 27;
     return previous;
 }

 public long next() {  // Marsaglia's XorShift
     for (;;) {
         var x = this.x;
         var newX = nextValue(x);
         if (varHandle.compareAndSet(this, x, newX)) {
             return newX * 2685821657736338717L;
         }
     }
 }
```

Steve Chen