**exo2:**

## exo2.1

**L’exécution de programme donne:**

```
la thread 0 d ébute: Thread id9
la thread 0 travaille : Thread id 9
la thread 0 termine : Thread id 9
la thread 1 d ébute: Thread id10
la thread 1 travaille : Thread id 10
la thread 1 termine : Thread id 10
la thread 2 d ébute: Thread id11
la thread 2 travaille : Thread id 11
la thread 2 termine : Thread id 11
la thread 3 d ébute: Thread id12
la thread 3 travaille : Thread id 12
la thread 3 termine : Thread id 12
la thread 4 d ébute: Thread id13
la thread 4 travaille : Thread id 13
la thread 4 termine : Thread id 13
la thread 5 d ébute: Thread id14
la thread 5 travaille : Thread id 14
la thread 5 termine : Thread id 14
la thread 6 d ébute: Thread id15
la thread 6 travaille : Thread id 15
la thread 6 termine : Thread id 15
la thread 8 d ébute: Thread id17
la thread 8 travaille : Thread id 17
la thread 8 termine : Thread id 17
la thread 7 d ébute: Thread id16
la thread 7 travaille : Thread id 16
la thread 7 termine : Thread id 16
la thread 9 d ébute: Thread id18
la thread 9 travaille : Thread id 18
la thread 9 termine : Thread id 18
```

## exo2.2

Dans ce  contexte multithread, on remarque que l’execution est séquentielle : Chaque thread attend qu’un autre  thread termine son travail pour qu’il commence ,à son tour, à travailler . Donc ilya a chaque fois qu’un seul thread qui accede a la variable  nextID (on a donc l’exclusion mutuelle) . Cette execution séquentielle est assurée grace a la methode synchronisée initialValue() de la classe interne ThreadLocal .

## Exo2.3

En concurrence , tous les threads d'un même processus partagent le même espace d’adressage. Donc, les données situées dans une variable statique ou globale sont exactement au même emplacement mémoire pour tous les threads, et correspondent donc à la même entité.

Pour éviter qu’une donnée soit partagée par plusieurs threads, La classe ThreadLocal permet d'encapsuler des données qui seront accessibles par tous les traitements exécutés dans le thread. La classe ThreadLocal implémente un mécanisme qui permet d'associer une donnée à un thread et de permettre son accès dans tous les traitements exécutés par le thread. Donc,Les variables sur la pile sont toutefois locales au thread, parce que chaque thread possède sa propre pile, distincte de celle des autres threads.