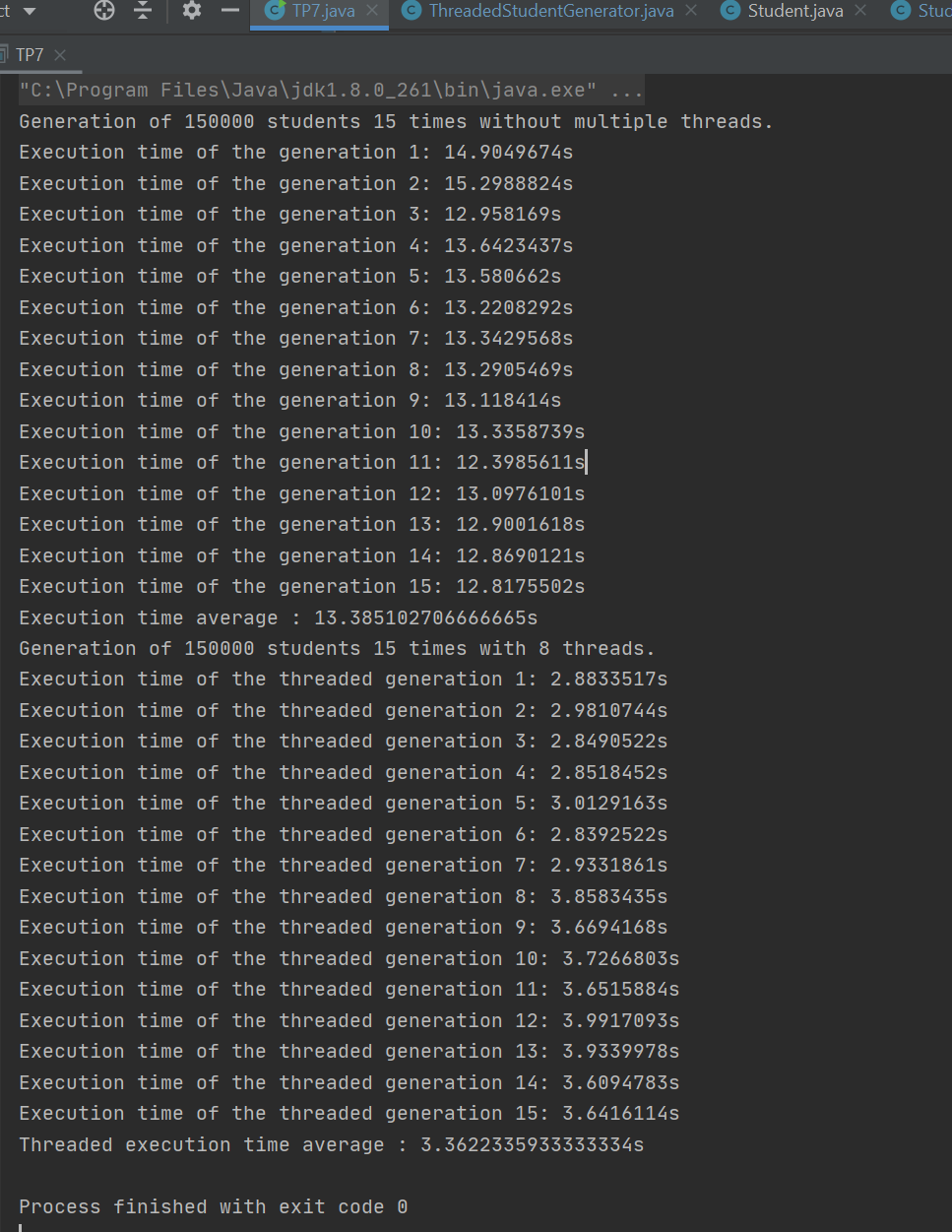
# Compte rendu TP7 Java

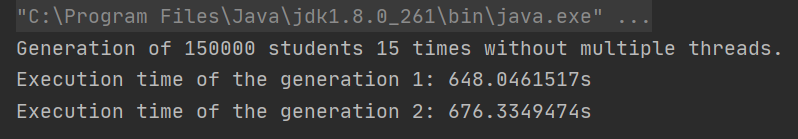
Fabien Simonet, ZZ3 F5, promotion 2021

J’ai utilisé un générateur de données appelé Javafaker. L’instanciation et l’utilisation d’un Faker est relativement longue ce qui permet d’allonger le temps d’exécution de la génération des étudiants afin d’avoir une comparaison plus intéressante entre le temps d’exécution des étudiants sans thread et avec threads.

Une fois la génération d’étudiants implémentées, les premiers tests ont donné les résultats ci-dessous :



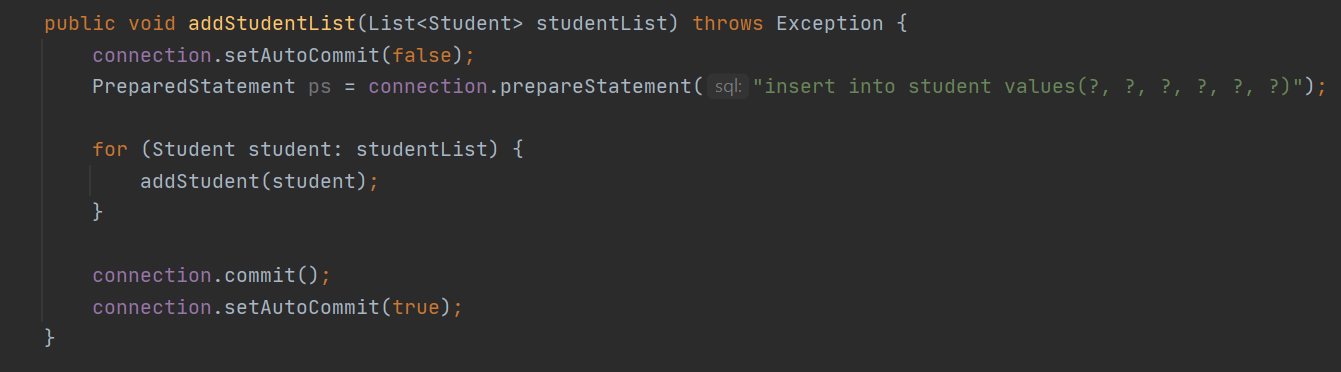
On peut voir que la génération d’étudiants avec 8 threads est environ 4 fois plus rapide que la génération non threadée. C’est moitié moins que ce à quoi on pourrait théoriquement s’attendre mais cela me parait tout de même cohérent.

J’ai ensuite ajouté une écriture en base de données, le résultat obtenu est le suivant :  


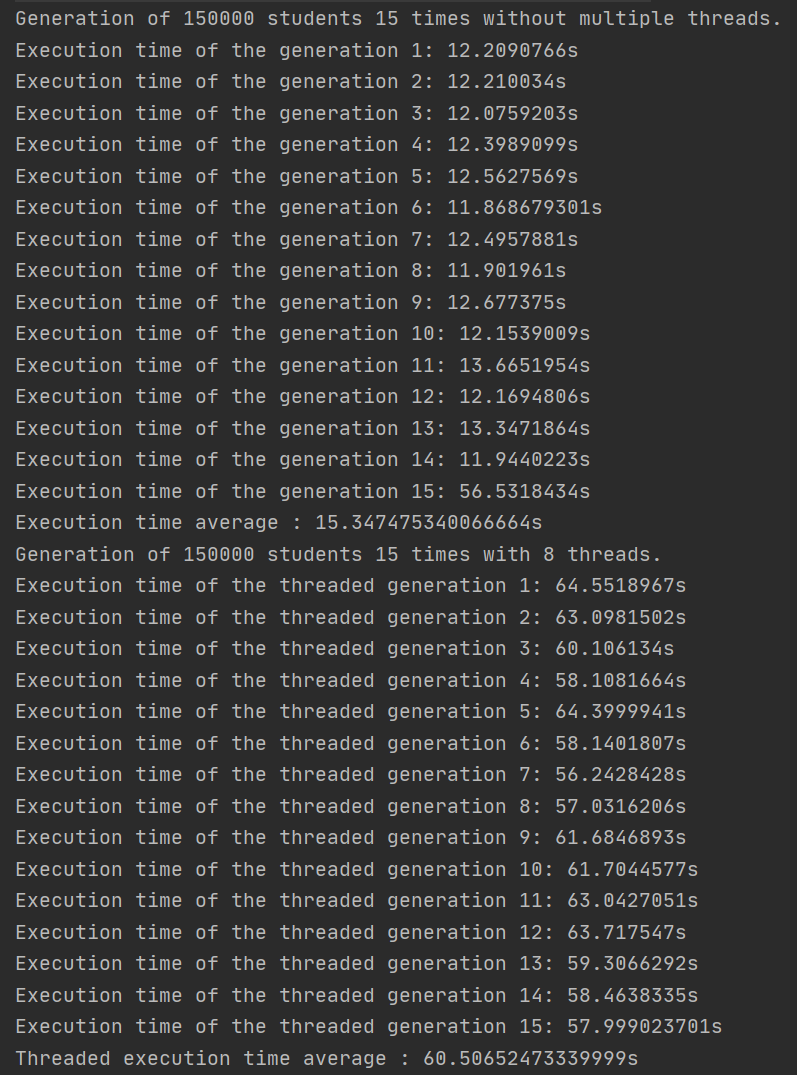
Le temps d’exécution est beaucoup trop long (11 minutes environ) pour la génération non threadée. Je n’ai pas testé la génération threadée car ces résultats me semblaient incohérents. L’écriture en base de données prend du temps mais pas au point de multiplier le temps d’exécution par 50.

En regardant mon code, je me suis rendu compte que je faisais une boucle for sur chacun de mes étudiants et que je les ajoutais en base un à un. J’ai fait quelques recherches et je me suis rendu compte que la connexion Java a une propriété autoCommit qui permet d’activer ou de désactiver le commit automatique. Par défaut, l’autoCommit est activé et se fait donc à chaque ajout d’étudiants. J’ai donc essayé de le désactiver avant d’exécuter mes requêtes d’ajout de chacun des étudiants, de faire le commit manuellement, puis de réactiver l’autoCommit une fois tout ça terminé. Cela permet de ne faire qu’un seul commit plutôt que 150 000 !

Le code ainsi modifié est le suivant :

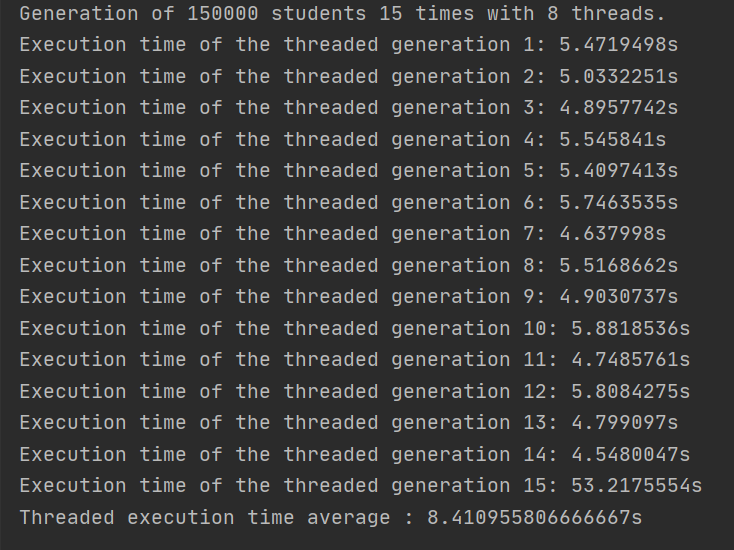


Les résultats obtenus sont les suivants :



On constate que désormais, l’ajout des étudiants en base de données est quasiment instantané, mise à part la dernière itération, mais que la génération threadée prend 20 fois plus de temps qu’auparavant, alors que je ne fait pas l’ajout en base de données sur la génération non threadée.

En commentant la génération non threadée et n’exécutant que la génération threadée (avec persistence), on obtient le résultat suivant :



Il semble donc se passer quelque chose sur la dernière itération. Par manque de temps, je ne suis pas allé plus loin dans mes recherches. Mais j’ai appris pas mal d’enseignements de ce tp. Le premier est qu’il y a des différences entre la théorie et la pratique concernant les threads et ce n’est pas parce qu’on à n threads que l’on divise forcément le temps par n. Je me suis aussi rendu compte que la parallélisation des tâches ne règle pas tous les problèmes et qu’il est parfois judicieux d’optimiser d’autres parties du codes plutôt que de paralléliser, comme par exemple les accès à la base de données. Enfin, je n’ai pas pris le temps d’essayer de paralléliser les accès à la base de données, mais je suis curieux de savoir s’il est possible de le faire, sachant que cela risque de poser des problèmes d’accès concurrent à la base en pagaille.

Dans le code que je vous envoie, j’ai volontairement réduit le nombre de générations d’étudiants (5 fois 150 000 au lieu de 15 fois 150 000) et commenté le code relatif aux écritures en base de données. Si vous souhaitez tester l’utilisation de la base, il suffit de décommenter le code.