RES - Labo 01 – Java I/O

1. Conditions de l'expérience

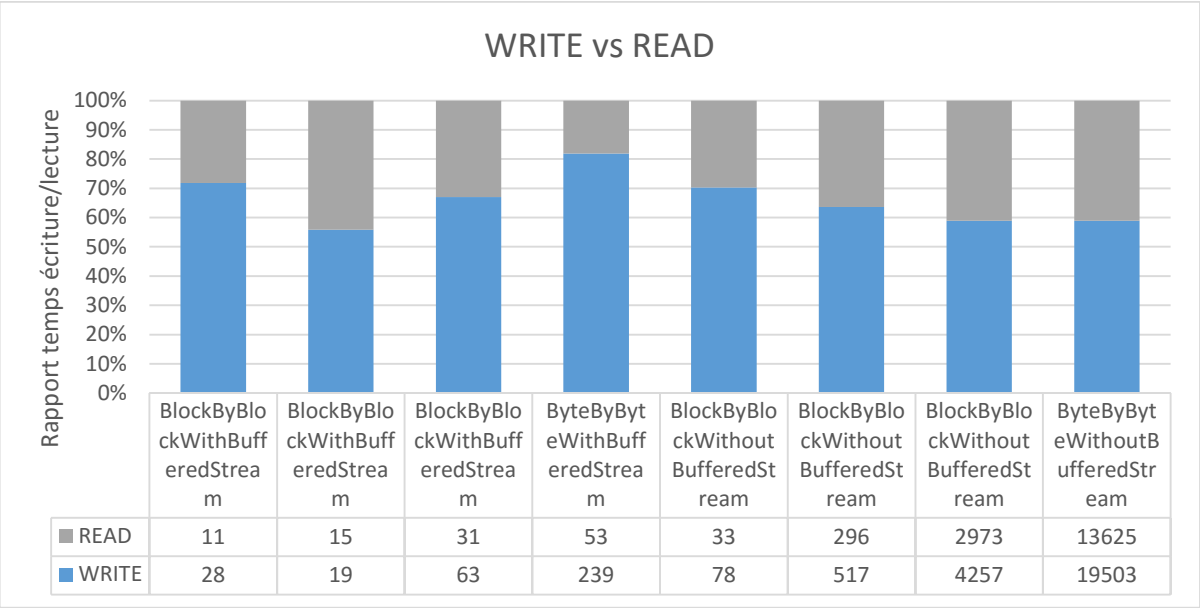
L'expérience consiste à comparer les performances d'exécution entre plusieurs types d'entrée/sortie de flux en JAVA (I/O stream).
Pour ce faire, un programme (qui nous a été fourni) s'occupe d'afficher dans la console les temps d'exécution nécessaires à l'écriture et à la lecture de données dans un fichier, au moyen de flux tamponnés ou non, tout en faisant varier la taille des blocs (plus le bloc est grand, plus on peut passer de données à la fois).

Pour ce laboratoire, l'expérience sera menée sur l'écriture (WRITE) et la lecture (READ) de données dans un fichier de 10MB, en utilisant des blocs de 0, 5, 50, ou 500 bytes, et en variant l'utilisation entre des flux tamponnés ou non.
Afin de présenter les mesures sous forme de graphes, les résultats d'exécution du programme ont été écrits dans un fichier CSV (Voir partie 3 sur la modification du code).

Les mesures ont été effectuées avec l'IDE Intellij sur Windows10 64 bits, processeur i-7 2.7GHz, disque dur 256GB SSD, RAM 8GB

2. Présentation des mesures et analyse

Ce premier graphique montre le rapport entre le temps d'écriture et de lecture pour les mêmes tailles de bloc. Le temps figurant dans le tableau sur l'axe des abscisses est en millisecondes.

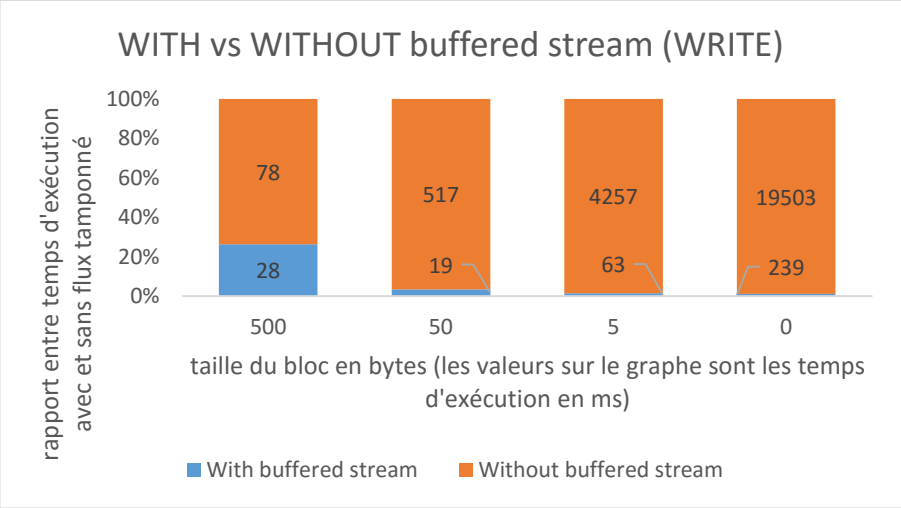


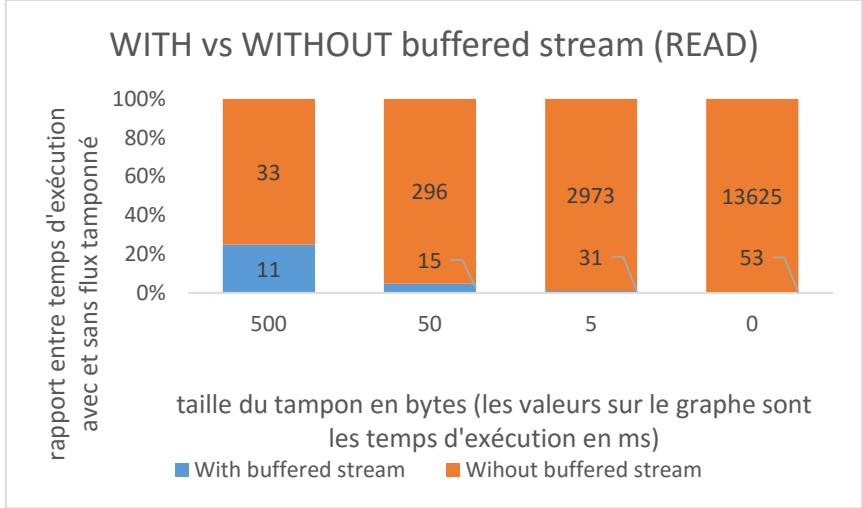
	BlockByBlockWithBufferedStream	BlockByBlockWithBufferedStream	BlockByBlockWithBufferedStream	ByteByByteWithBufferedStream	BlockByBlockWithoutBufferedStream	BlockByBlockWithoutBufferedStream	BlockByBlockWithoutBufferedStream	ByteByByteWithoutBufferedStream
READ	11	15	31	53	33	296	2973	13625
WRITE	28	19	63	239	78	517	4257	19503

Analyse
Les temps d'écriture sont toujours supérieurs aux temps de lecture, ceci provient du fait qu'il y a une ou deux lignes de code assembleur supplémentaires à effectuer lors d'une écriture dans un fichier.

Il n'y a pas vraiment de rapport constant entre l'écriture et la lecture mais on peut observer, à vue d'œil, une moyenne de 68% avec une marge d'erreur moyenne de ± 7%.

Ce deuxième graphique montre l'intérêt d'utiliser les flux tamponnés





Analyse

L'utilisation de flux tamponnés (« buffered stream ») a une importance significative ! On constate sur ce graphe l'énorme différence de temps d'exécution entre des flux avec et sans tampons. Les flux tamponnés sont bien plus performants car ils n'effectuent que très rarement des appels systèmes pour écrire ou lire dans un fichier (les appels sont réalisés lorsque le tampon est plein pour l'écriture, et vide pour la lecture).
Cependant, plus les blocs sont volumineux, moins la différence de temps est grande. Dans ce cas, la taille du bloc joue un peu le rôle de tampon.

3. Explications sur la modification du code

Pour pouvoir écrire dans un fichier CSV, j'ai suivi le webcast youtube de notre professeur M. Liechti et est appliqué le même design à mon code.
L'écriture se décompose de la manière suivante :

1. Ajout d'un serializer et d'un recorder (FileRecorder) comme attributs privés à la classe BufferedIOBenchmark.
2. Initialisation du recorder dans main pour permettre l'écriture dans un PrintStream après avoir sérialisé les données saisies au moyen de la classe MyExperimentData.
3. Saisie des données et enregistrement de celles-ci (au format csv).
4. Fermeture du recorder à la fin du programme.

Les données sont stockées dans une Map<int, String> (Java.util). Dans ce labo, les clés sont identifiées ainsi : 0=operation, 1=Strategy, 2=blockSize, 3=taille fichier en byte, 4=durée.

Pour pouvoir rendre le code plus générique, j'ai créé des interfaces pour que les différents éléments communiquent au travers de celles-ci, selon les bons conseils de la vidéo.