

Java Performance - TP4

November 13, 2018

Thomas COUCHOUD
thomas.couchoud@etu.univ-tours.fr
Victor COLEAU
victor.coleau@etu.univ-tours.fr



Chapter 1

Introduction

Chapter 2

Exercices

2.1 Exercice 1

On remarque que beaucoup de types utilisés des classes (Float, Integer, ...), il serait donc peut-être plus efficace d'utiliser des types primitifs directement afin d'éviter d'inutiles opérations d'unboxing/boxing.

De plus nous avons changé les "k = k + 1" en "k++".

Après des mini benchmarks nous obtenons la sortie suivante:

Listing 2.1 – benchmark.txt

	Benchmark	(i)	Mode	Cnt	Score	Error	Units
1	Exercice1.test	0	thrpt	10	70006.601	+= 1883.562	ops/s
2	Exercice1.test	1	thrpt	10	71369.058	+= 1905.356	ops/s
3	Exercice1.testA	0	thrpt	10	457780.134	+= 6152.443	ops/s
4	Exercice1.testA	1	thrpt	10	467063.858	+= 7418.504	ops/s

Avec nos améliorations on peut constater que le score est multiplié par environ 5. Cela est probablement dû aux changements des types plus qu'à la transformation des incréments.

2.2 Exercice 2

La première remarque est que dans le calcul d'une valeur de fibonacci, le calcul des deux précédentes est nécessaire. Or dans l'implémentation donnée fibonacci(i-1) et fibonacci(i-2) sont indépendantes alors que fibonacci(i-1) utilise et donc recalcule lui-même fibonacci(i-2). Ce dernier calcul est donc effectué au moins 2 fois, ce qui est inutile et peut être très long sur des grandes valeurs.

Nous allons donc mettre en place un système de cache des valeurs au travers d'une map (fibonacciA).

De plus, étant donné que l'on demande la valeur de fibonacci(43), nous pouvons aussi renvoyer la valeur directement en l'ayant calculée auparavant (fibonacciB).

Listing 2.2 – benchmark.txt

	Benchmark	(i)	Mode	Cnt	Score	Error	Units
1	Exercice2.test	43	thrpt	10	~= 0		ops/s
2	Exercice2.testA	43	thrpt	10	23459059.162	+= 1869717.291	ops/s

L'amélioration est ici flagrante avec le cache, et encore plus avec la valeur renvoyée directement.

2.3 Exercice 3

2.3.1 Exercice 3a

2.3.2 Exercice 3b

2.4 Exercice 4

La première chose que nous remarquons est que l'on fait une boucle for pour ajouter tous les éléments d'un tableau.

La première méthode consistant à utiliser le .addAll de la liste (exercice4A). Cependant cela nous a forcé à convertir pleins de types ce qui ne va probablement pas être optimal.

Une deuxième méthode a été d'utiliser directement les streams avec un flatMap, cependant encore une fois cela implique de faire des conversions de type.

Notre dernière méthode consiste à d'abord calculer la taille du tableau final, puis le remplir directement. Cela évite d'ajouter une fois dans une liste, puis d'ajouter dans un tableau.

2.5 Exercice 5

Ici la réflexivité est utilisée sur un objet dont on connaît déjà le type et qui contient déjà la méthode à appeler. Tout cet enchainement est inutile car nous pouvons appeler le .getName() directement sur notre objet. Cela va nous permettre de s'affranchir de la recherche de la méthode et invocation de cette dernière.

Listing 2.3 – benchmark.txt

	Benchmark	(i)	Mode	Cnt	Score	Error	Units
1	Exercice5.test	Thomas	thrpt	10	1257489.509	+= 68643.635	ops/s
2	Exercice5.test	Clement	thrpt	10	1261393.467	+= 55132.028	ops/s
3	Exercice5.test	Louis	thrpt	10	1220180.478	+= 185204.978	ops/s
4	Exercice5.testA	Thomas	thrpt	10	57787733.041	+= 329682.747	ops/s
5	Exercice5.testA	Clement	thrpt	10	57331680.503	+= 1011760.686	ops/s
6	Exercice5.testA	Louis	thrpt	10	57808907.426	+= 332110.507	ops/s
7							

Encore une fois on remarque que ces changements ont grandement augmenté les performances.

Chapter 3

Conclusion