Java 8 Benchmarks

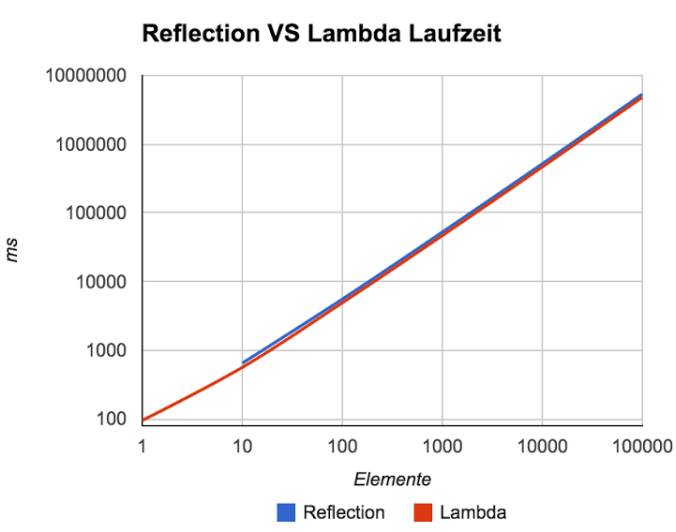
Durchführung

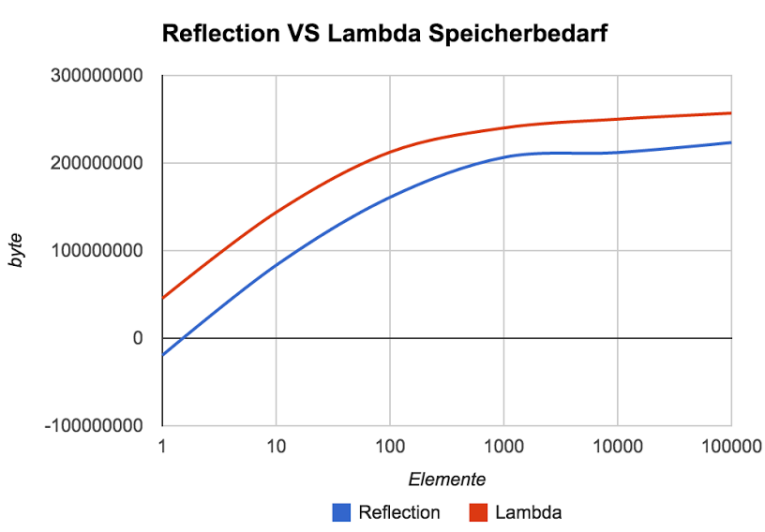
Die Benchmarks wurden auf Basis des Micro-Frameworks von Kent Beck und Bernd Kahlbrandt durchgehführt. Dieses wurde um Funktionalität zur Speichermessung erweitert.

In den Testobjekten werden Manipulationen (in Form von .toUpperCase()) auf String-Objekten (aus SGB-Words[1]) durchgeführt.

Ergebnisse

Reflection vs Lambda-Ausdrücke

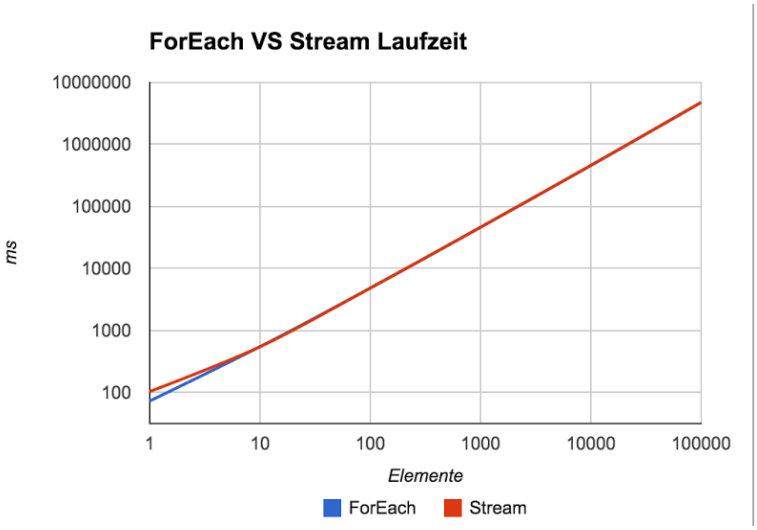


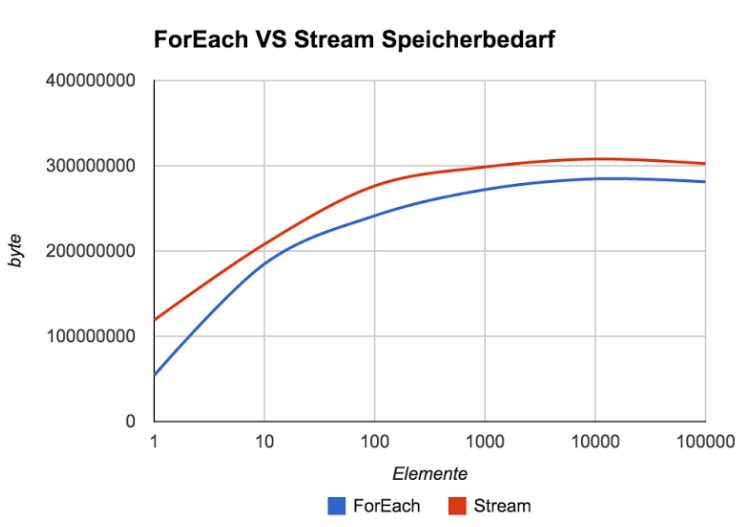


Es ist zu erkennen, dass Methodenaufrufe per Reflection und Aufrufe per Lambda-Ausdrücke in etwa gleich viel Zeit in Anspruch nehmen. Wie aus dem Graphen ersichtlich ist, nimmt die Verwendung von Lambda-Ausdrücken eine fixe Menge von zusätzlichem Speicher in Anspruch. Dieser zusätzliche Speicherbedarf hängt nicht von der Anzahl der Elemente ab. Vermutlich verbraucht das Lambda-Konstrukt an sich mehr Speicher, als Reflection.

[1] SGB-Words eignen sich gut zur Verwendung in Benchmarks, da alle generierten Wörter dieselbe Länge haben.

ForEach vs Stream

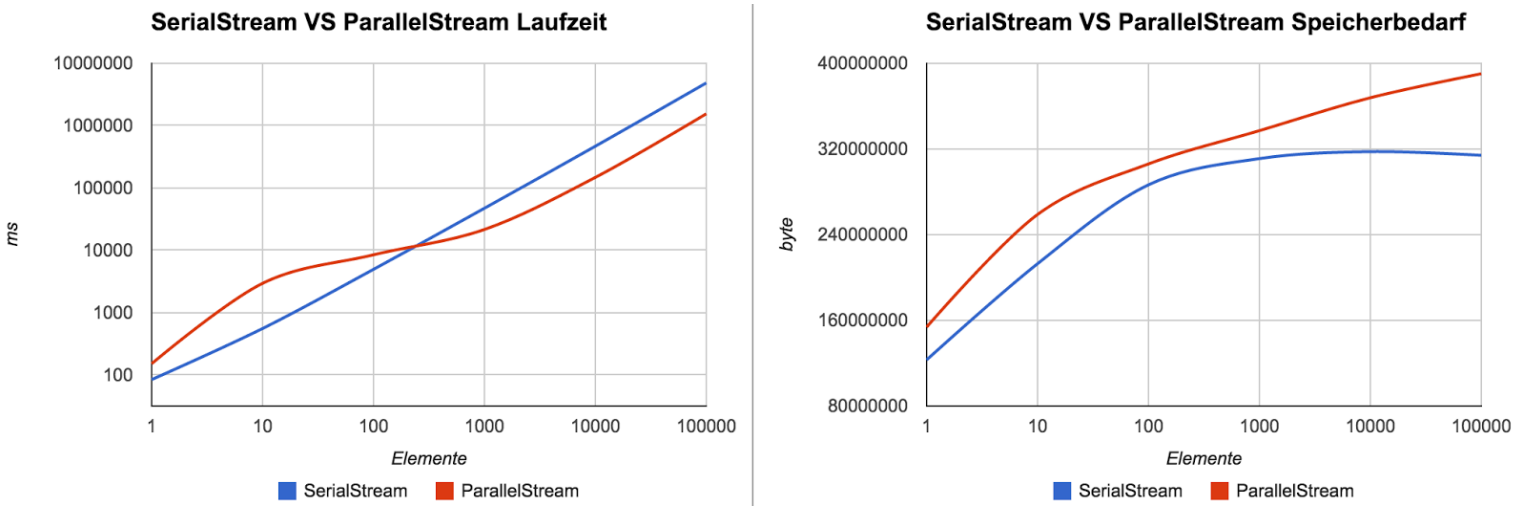




In diesem Benchmark wurde der klassische for-Loop mit Iteratoren mit Streams verglichen. Der Initialisierungs-Overhead ist bei Streams zu Beginn sehr viel höher, als bei for-Schleifen. Dies ist vermutlich darauf zurückzuführen, dass for-Schleifen zu Beginn der Iterationen lediglich den Iterator initialisieren müssen, während Streams sowohl das Stream-Objekt, als auch das aus dem Lamda-Ausdruck resultierende Objekt instanziieren müssen.

Dies nimmt eine fixe Menge Speicher und feste Laufzeit in Anspruch.

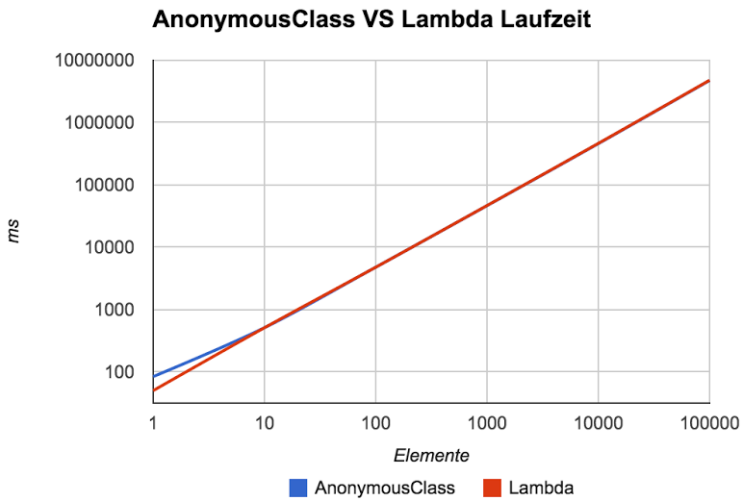
SerialStream VS ParallelStream

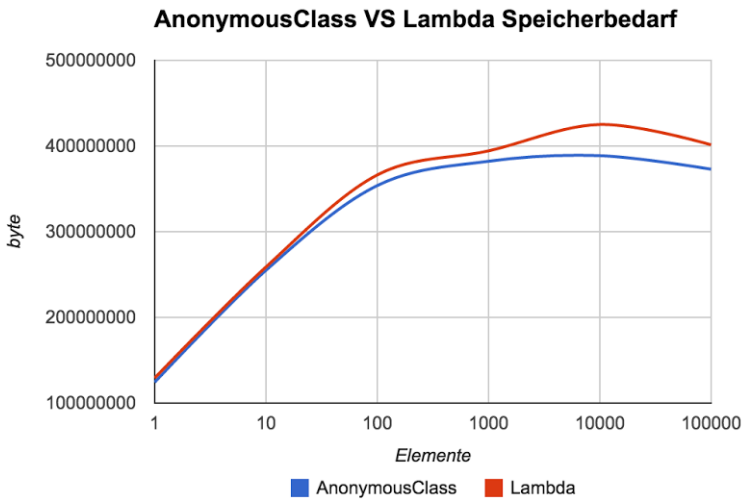


Ein ParallelStream erzeugt bei der Berechnung mehrere Threads, was sich bei wenigen Elementen negativ auf Laufzeit und Speicherverbrauch auswirkt. Bei ausreichend großen Mengen (in unserem Fall[2] mehr als 200 Elemente), wird die Laufzeit um ungefähr ⅔ verkürzt. Allerdings wird bei verringerter Laufzeit wesentlich mehr Speicher in Anspruch genommen (etwa ⅓ mehr). Da wir also nur ⅓ mehr Speicher bei ⅔ weniger Laufzeit benötigen, arbeiten wir bei ParallelStreams speichereffizienter als bei SerialStreams.

[2] Getestet auf einem Quadcore-Prozessor mit 4 Ghz Taktung.

Anonyme Klassen VS Lambda-Ausdrücke





Bei anonymen Klassen lässt sich eine längere Initialisierungszeit gegenüber Lambda-Ausdrücken beobachten. Langfristig nehmen sich beide Lösungen aber nichts. Der Speicherbedarf ist bei steigender Anzahl zu verarbeitender Elemende bei Lambda-Ausdrücken höher als bei anonymen Klassen.

Fazit

Bei allen durchgeführten Tests haben wir festgestellt, dass sich die gegenübergestellten Lösungen (mit Ausnahme von ParallelStream VS SerialStream) nicht viel nehmen. Es können also ohne Bedenken die neuen Features von Java 8 zur Verbesserung der Lesbarkeit verwendet werden, sofern die Einsatzumgebung der Anwendung nicht besonders speicherkritisch (wie z.B. bei Embedded-Systems) ist.

Der Einsatz von ParallelStreams lohnt sich bei parallelisierbaren Aktionen bei der Entwicklung ohne Mehraufwand sehr.