где $M(t,x,y,z)\in G$; $t=t_1$, $x=x_2$, $y=x_2$, $z=x_4$; $\tau=(\tau_1,\tau_2,\tau_3,\tau_4)\in\partial G$; $\omega_n=\dfrac{2\pi^{\frac{n}{2}}}{\Gamma\left(\dfrac{n}{2}\right)}$, $n=3$;

под знаком интеграла интеграла $f=f(\tau)$; α_1, α_2, α_3, α_4 – направляющие косинусы внешней нормали к ∂G; $p_j=\dfrac{\partial p(\tau)}{\partial\tau_j}$; $\varphi^j=\dfrac{\tau_j-x_j}{r^4}$ ($j=1,2,3,4$);

$r=\sqrt{(\tau_1-x_1)^2+(\tau_2-x_2)^2+(\tau_3-x_3)^2+(\tau_4-x_4)^2}$.

При этом предполагается, что поверхность ∂G удовлетворяет условиям, при которых имеет место формула Остроградского.

Рассмотрим следующую краевую задачу.

Задача. Найти значение кватернионной функции

$f=f_1(t,x,y,z)+f_2(t,x,y,z)i+f_3(t,x,y,z)j+f_4(t,x,y,z)k$

внутри замкнутой поверхности ∂G (∂G – граница области $G\subset D$) по её значениям на поверхности области G, если функция f является F-моногенной по функции $p=t+\varepsilon(x+y+z)$, $\varepsilon=\lambda_1 i+\lambda_2 j+\lambda_3 k$, $\lambda_1^2+\lambda_2^2+\lambda_3^2=\dfrac{1}{3}$, а поверхность ∂G удовлетворяет условиям, при которых возможно применение формулы Остроградского.

Заключение. Формула (3) и дает возможность решить сформулированную выше краевую задачу.

1. Гусев, В. А. О кватернионных функциях, моногенных в смысле В. С. Фёдорова / В. А. Гусев // Успехи математических наук. – 1965. – Т. 20. – Вып. 1(121).– С. 203–208.

2. Фёдоров, В. С. Основные свойства обощенных моногенных функций / В. С. Фёдоров // Известия вузов. Математика. – 1958.– № 6.– С. 257–265.

3. Стэльмашук, М. Т. Аб інтэгральным выяўленні кватерніённых F-манагенных функцый аднаго класа / М. Т. Стэльмашук, У. А. Шылінец // Весці БДПУ. Сер. 3.– 2005.– № 2.– С. 8–10.

4. Стэльмашук, М. Т. Рашэнне краявой задачы для кватэрніённых функцый чатырох рэчаісных зменных / М. Т. Стэльмашук, У. А. Шылінец, Г. Ф. Падабед // Весці БДПУ. Сер. 3.– 2006.– № 1.– С. 12–14.

5. Стэльмашук, М. Т. Аб кватэрніённых манагенных у сэнсе У. С. Фёдарава функцыях / М. Т. Стэльмашук, У. А. Шылінец, Г. А. Андрэева // Весці БДПУ. Сер. 3.– 2010.– № 1.– С. 11–13.

6. Фёдоров, В. С. Об одном обобщении интеграла типа Коши в многомерном пространстве / В. С. Фёдоров // Известия вузов. Математика. – 1957. – № 1.– с.227–233.

ИССЛЕДОВАНИЕ ПРОИЗВОДИТЕЛЬНОСТИ JAVA STREAM API И ГИБКОСТИ ПОДДЕРЖКИ КОДА, ЕГО ИСПОЛЬЗУЮЩЕГО

Грицкевич Н.С.,

студент 2-го курса ВГУ имени П.М. Машерова, г. Витебск, Республика Беларусь

Научный руководитель – ***Ермоченко С.А.,*** *канд. физ.-мат. наук, доцент*

Ключевые слова. Java коллекции, Java Stream API, итераторы, производительность, время работы

Keywords. Java collections, Java Stream API, iterators, performance, work time

Зачастую при написании кода возникает необходимость выбора подхода к реализации поставленной задачи, требующей обработки некоторой последовательности данных. Есть несколько вариантов реализации: через циклы (for, foreach, while, do while), с помощью методов, которые предоставляются коллекциями (Collections), с помощью Iterator-ов или Stream API [1]. При разработке большую роль играет не только скорость выполнения

алгоритма, но и простота читаемости кода, а также его компактность, что влияет на гибкость поддержки кода.

Целью данной работы является исследование времени работы алгоритмов, а также гибкости поддержки кода.

Материалы и методы. Для анализа использовалась программа IntelliJ IDEA 19.2.2 и Java 1.8. Были применены, принципы объектно-ориентированного программирования и накопленный опыт в этой области. Использовались следующие методы исследования: теоретические – изучение литературных источников по теме исследования, эмпирические – проведение экспериментов по замеру производительности отдельных реализаций тестовых задач.

Результаты и их обсуждение. В ходе исследования были проведены исследования времени работы некоторых из методов Stream API с коллекциями, а также реализация аналогичных задач через циклы и Iterator-ы. При исследовании была создана коллекция List элементов типа Integer. В качестве реализации интерфейса List был использован класс ArrayList. При выборе другой реализации интерфейса List, результаты экспериментов могут быть другими. Для проведения замеров производительности данная коллекция была заполнена случайными целыми числами в диапазоне значений от 0 до 100 и в количестве от 100 тыс. элементов до 100 млн. элементов.

Для оценки производительности решались следующие тестовые задачи:

1. Нахождения максимального элемента в коллекции. Выполнялось для 100 000 и 1 000 000 элементов,
2. Нахождения всех элементов меньше 93 с последующим увеличением на единицу.
3. Удаление дубликатов из коллекции.
4. Сумма всех элементов коллекции.
5. Среднее арифметическое всех элементов коллекции.
6. Проверка, все ли элементы коллекции – четные числа.
7. Сортировка коллекции по убыванию с нахождением суммы всех элементов.

Далее приведём результаты тестирования с учётом времени работы (t) программы (значения в миллисекундах), а также анализу относительной производительности (p) программы (значения в процентах). Под абсолютной производительностью будем понимать отношение количества обрабатываемых элементов к времени обработки. Относительную производительность циклов будем принимать за 100%.

В таблице 1 приведены результаты исследований для реализации алгоритма через Stream API (S), через циклы (L) и через Iterator-ы (I). Для некоторых задач использовались лишь циклы, так как реализация через Iterator не возможна из-за функциональных ограничений. Значение ‘н/д’ означает, что проведение тестирования при таких условиях либо не доступно (из-за слишком большого времени работы или из-за ограничений на используемую память), либо не имеет смысла (время работы слишком мало или слишком незначительно отличается от аналогичного теста на другом количестве элементов):

Таблица 1. – Результаты исследований для реализации алгоритма через Stream API (S), через циклы (L) и через Iterator-ы (I).

Количество элементов	**Реализация**	**Номер задачи**													
		1		**2**		**3**		**4**		**5**		**6**		**7**	
		t, мс	p, %	t, мс	p, %	t, мс	p, %	t, мс	p, %	t, мс	p, %	t, мс	p, %	t, мс	p, %
150 тыс.	L	н/д		13	100	23924	100	4	100	5	100	4	100	66	100
	I			–	–	–	–	6	166	10	50	2	200	–	–
	S			7	185	9	2142800	5	80	9	55	3	133	26	253

200 тыс.	L	8	100	н/д		н/д				н/д				н/д			
	I	–	–														
	S	6	175														
250 тыс.	L	н/д		н/д		66507	100			н/д		251	100	249	100		
	I					–	–					246	102	2	12450	н/д	
	S					9	740700					128	196	207	120		
100 млн.	L	651	100	2056	100					39	100					19953	100
	I	–	–	–	–	н/д				470	8	н/д		н/д		–	–
	S	415	164	251	819					668	5					1318	1513

Проведённые тесты показали в целом высокую производительность Stream API. В отдельных задачах, таких как суммирование элементов коллекции, Stream API показывает результат значительно хуже, чем реализация через циклы, но сопоставимый с реализацией через Iterator. Это обусловлено тем, что сама по себе задача достаточно простая, и в абсолютных значениях даже на 100 млн. элементов Stream API работает менее 1с, но из-за накладных расходов на создание объектов и вызов методов и Stream API, и Iterator показывает результат хуже, чем циклы. Но в более сложных задачах, особенно требующих нескольких этапов обработки, например, удаление дубликатов или сортировка с нахождением суммы, Stream API показывают скорость в несколько раз превосходящий скорость работы цикла. Особенно это заметно на задаче удаления дубликатов. Хотя стоит отметить, что сравнение производилось с реализацией этой задачи с помощью циклов, в которой не использовались никакие оптимизации.

Анализируя читаемость и компактность кода при использовании циклов, Iterator и Stream API, можно отметить, что код цикла и работа с Iterator требует около 2-4 строк кода, содержащих вспомогательный «рутинный» код, затрудняющий восприятие. При решении сложных задач с нетривиальными условиями цикла и сложной логикой в теле цикла такой «рутинный» код практически не мешает воспринимать основную логику решения задачи, но в маленьких типовых задачах это не так. В отличие от использования циклов и Iterator код, использующий Stream API, гораздо компактнее и практически лишён вспомогательных конструкций. Однако для восприятия этого кода программисту потребуется знания нового синтаксиса Java, такого как лямбда-выражения и ссылки на методы, что может вызвать некоторые трудности у начинающих программистов.

Заключение. По итогам тестирования в большинстве случаев Stream показывает производительность и время работы намного больше, так как Stream очень быстро работает с большими объемами данных. Таким образом Stream предоставляет гибкость поддержки кода, а также более высокую компактность, что делает код более читаемый. Данное исследование может поможет разработчику в выборе реализации для решения поставленной задачи.

1. Гослинг Дж. и др. Язык программирования Java SE 8. – Москва: Вильямс, 2015. – 672 с.

ИСПОЛЬЗОВАНИЕ МАТЕМАТИЧЕСКИХ МОДЕЛЕЙ ПРИ ИЗУЧЕНИИ МНОГОГРАННИКОВ В УСЛОВИЯХ ДИСТАНЦИОННОГО ОБУЧЕНИЯ

Дмитриева О.А.,
магистрант ПсковГУ, г. Псков, Российская Федерация
Научный руководитель – ***Медведева И.Н.,*** *канд. физ.-мат. наук, доцент*

Ключевые слова. математическая модель, звездчатый многогранник, дистанционное обучение, геймификация.

Key words. mathematical model, stellate polyhedron, distance education, gamification.