Санкт-Петербургский политехнический университет Петра Великого

Высшая школа интеллектуальных систем и суперкомпьютерных технологий.

**Курсовая работа**

**Дисциплина**: Операционные системы

**Тема**: Разработка многопоточной программы

Выполнила студентка гр. 3530901/80203 Рубша А.И.

(подпись)

Преподаватель Егорова И. С.

(подпись)

“\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

Санкт-Петербург

2020

# **Формулировка задания:**

В качестве курсовой работы мною была выбрана разработка программы, реализующей определение частоты употребления каждого слова в тексте, написанным пользователем (на языке Java).

Весь код, расположенный ниже, прокомментирован и лежит на гитхабе по ссылке:

**Описание алгоритма:**

В случае многопоточности – задача разбивалась на несколько подзадач, каждую из которых выполнял отдельный поток. После чего, все результаты складывались в один, в результате чего мы получали итоговый список частот употребления слов в тексте (предложении).

Основа алгоритма (к примеру, без многопоточности) заключалась в следующем:

Компьютер просит пользователя ввести предложение (строку, текст, набор слов). После того, как пользователь вводит нужное, программа заносит полученную строку в отдельный массив, при это переводя все слова в нижний регистр и с помощью регексов сортирую проходимость слов, то есть все запятые, точки, пробелы, и другие знаки препинания будут убраны и на выходе мы получим массив слов. (К примеру, в моей программе слова ПРИВЕТ и привет – будут одинаковыми).

Также моя программа создает hashSet, в котором будут храниться только уникальные слова, в дальнейшем этот список нам понадобится для подсчета каждого уникального слова. Далее, проходя по массиву всех слов и hashSet-у, программа каждый раз встречая определенное слово в списке, будет прибавлять 1 к его количеству.

Таким образом, на выходе мы получаем 2 массива – снова тот же hashSet и новый массив arr, представляющий собой массив чисел (каждое число равно частоте употребления каждого слова в общем списке).

Затем, когда весь подсчет будет окончен, мы будем выводить на консоль предложение, содержащее информацию о каждом из слове и о его частоте.

Класс Main содержит 2 метода – withoutThreads() и withThreads().

# **Однопоточная реализация:**

static void withoutThreads() throws IOException {  
 BufferedReader br = new BufferedReader(new InputStreamReader(System.*in*));  
 System.*out*.println("Пожалуйста, введите ваше предложение: ");  
 String st = br.readLine();  
 String[] words = st.toLowerCase().split("[\\s,. :\"';%№!?]+");//массив всех слов разделенных  
 ArrayList<String> arrayList = new ArrayList<>(Arrays.*asList*(words)); //лист в который мы будем пихать слова которые уже подсчитали  
 HashSet<String> hashSet = new HashSet<>(Arrays.*asList*(words));  
 final Integer[] arr = new Integer[hashSet.size()];  
 for (int i = 0; i < hashSet.size(); i++) {  
 arr[i] = 0;  
 }  
 long allTime = 0;  
 AtomicInteger count = new AtomicInteger();  
 long start = System.*nanoTime*();  
 Iterator<String> p = hashSet.iterator();  
 long time = allTime;  
 for (int t = 0 ; t < hashSet.size(); t++) {  
 String word = p.next();  
 for (int j = 0; j < arrayList.size(); j ++) {  
 if (words[j].equals(word)) {  
 arr[t]++;  
 count.getAndIncrement();  
 words[j] = "-";  
 }  
 }  
 if (count.intValue() == arrayList.size()) {  
 System.*out*.println(count.intValue() + " " + arrayList.size());  
 long finish = System.*nanoTime*();  
 time += (finish - start);  
 System.*out*.println("Время затраченное без потоков(мс) = " + time);  
 break;  
 }  
 }  
 if (count.intValue() == arrayList.size()){  
 Iterator<String> iter = hashSet.iterator();  
 for (int j = 0; j < hashSet.size(); j++) {  
 String word = iter.next();  
 System.*out*.println("Частота употребления слова - " + word + " = " + arr[j]);  
 }  
 }  
}

# **Многопоточная реализация:**

static void withThreads() throws IOException { //  
 int THREADS = 11;  
 BufferedReader br = new BufferedReader(new InputStreamReader(System.*in*));  
 System.*out*.println("Пожалуйста, введите ваше предложение: ");  
 String st = br.readLine();  
 String[] words = st.toLowerCase().split("[\\s,. :\"';%№!?]+");//массив всех слов разделенных  
 ArrayList<String> arrayList = new ArrayList<>(Arrays.*asList*(words)); //лист в который мы будем пихать слова которые уже подсчитали  
 HashSet<String> hashSet = new HashSet<>(Arrays.*asList*(words));  
 final Integer[] arr = new Integer[hashSet.size()];  
 for (int i = 0; i < hashSet.size(); i++) {  
 arr[i] = 0;  
 }  
 long allTime = 0;  
 AtomicInteger count = new AtomicInteger();  
 for (int i = 0; i < THREADS; i++) {  
 long start = System.*nanoTime*();  
 int finalI = i;  
 Thread thread = new Thread(() -> {  
 Iterator<String> p = hashSet.iterator();  
 long time = allTime;  
 for (int t = 0 ; t < hashSet.size(); t++) {  
 String word = p.next();  
 for (int j = finalI; j < arrayList.size(); j += THREADS) {  
 if (words[j].equals(word)) {  
 arr[t]++;  
 count.getAndIncrement();  
 words[j] = "-";  
 }  
 }  
 if (count.intValue() == arrayList.size()) {  
 System.*out*.println(count.intValue() + " " + arrayList.size());  
 long finish = System.*nanoTime*();  
 time += (finish - start);  
 System.*out*.println("Время затраченное с потоками(мс) = " + time);  
 break;  
 }  
 }  
  
 if (count.intValue() == arrayList.size()){  
 Iterator<String> iter = hashSet.iterator();  
 for (int j = 0; j < hashSet.size(); j++) {  
 String word = iter.next();  
 System.*out*.println("Частота употребления слова - " + word + " = " + arr[j]);  
 }  
 }  
 });  
 thread.start();  
 }  
 }

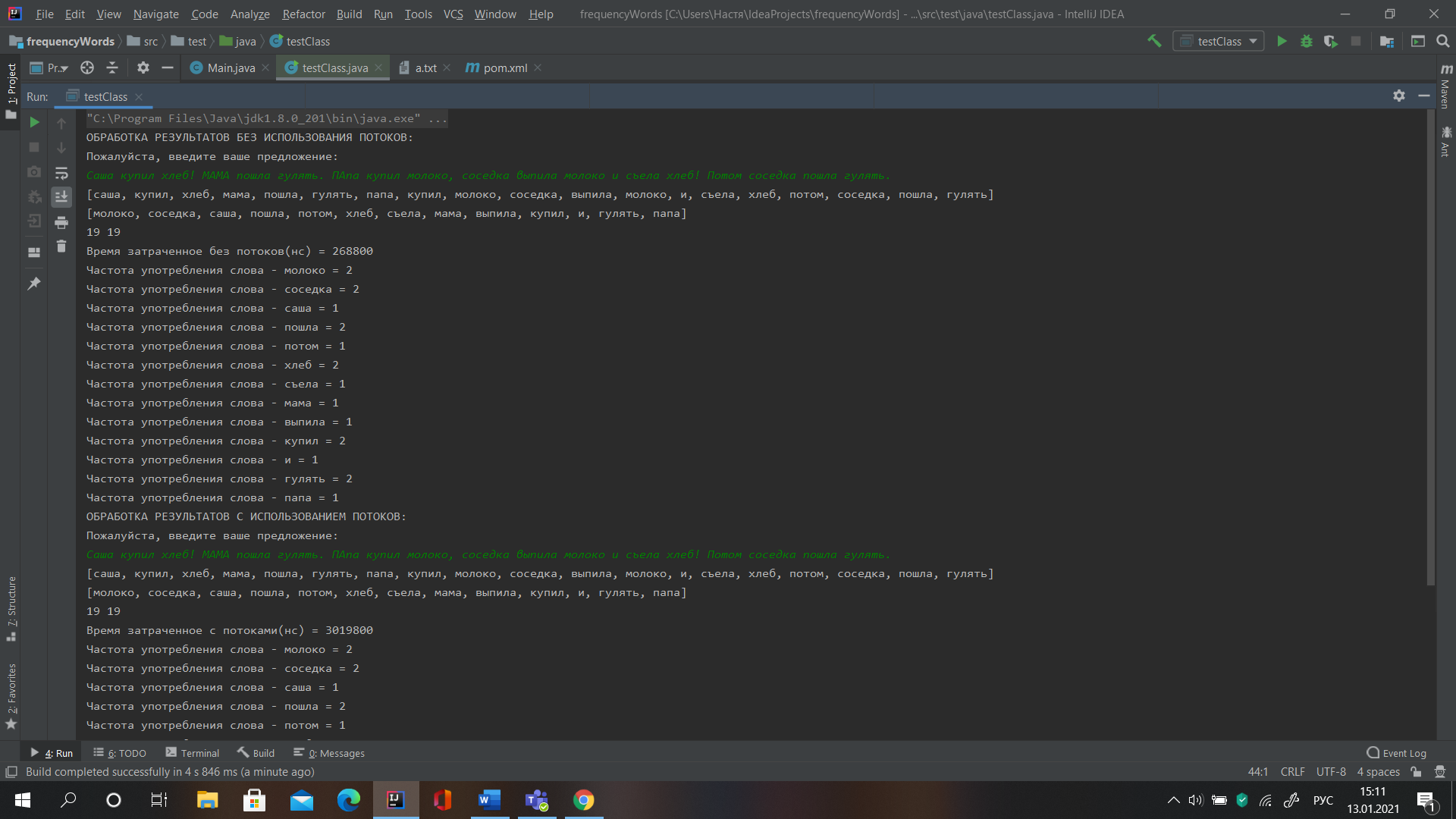
# **Сравнение реализаций:**

Для тестирования однопоточной и многопоточной сортировок, а также сравнения их скорости, был написан класс testClass.

Все тестирование было сделано вручную, все результаты работы программ будут представлены ниже, в разделе Результаты работы.

public class testClass {  
 public static void main(String[] args) throws IOException {  
 Main main = new Main();  
 System.*out*.println("ОБРАБОТКА РЕЗУЛЬТАТОВ БЕЗ ИСПОЛЬЗОВАНИЯ ПОТОКОВ: ");  
 main.*withoutThreads*();  
 System.*out*.println("ОБРАБОТКА РЕЗУЛЬТАТОВ С ИСПОЛЬЗОВАНИЕМ ПОТОКОВ: ");  
 main.*withThreads*();  
 //Все тесты, проведенные в данном классе, были зафиксированы в отчете  
 //То бишь была изменена несколько раз размерность предложений. В зависимости от этого было также посчитано время выполенния  
 //программ как с использованием потоков, так и без.  
 }  
}

**Результаты работы:**

Протестируем на массиве из маленького количества элементов (19):

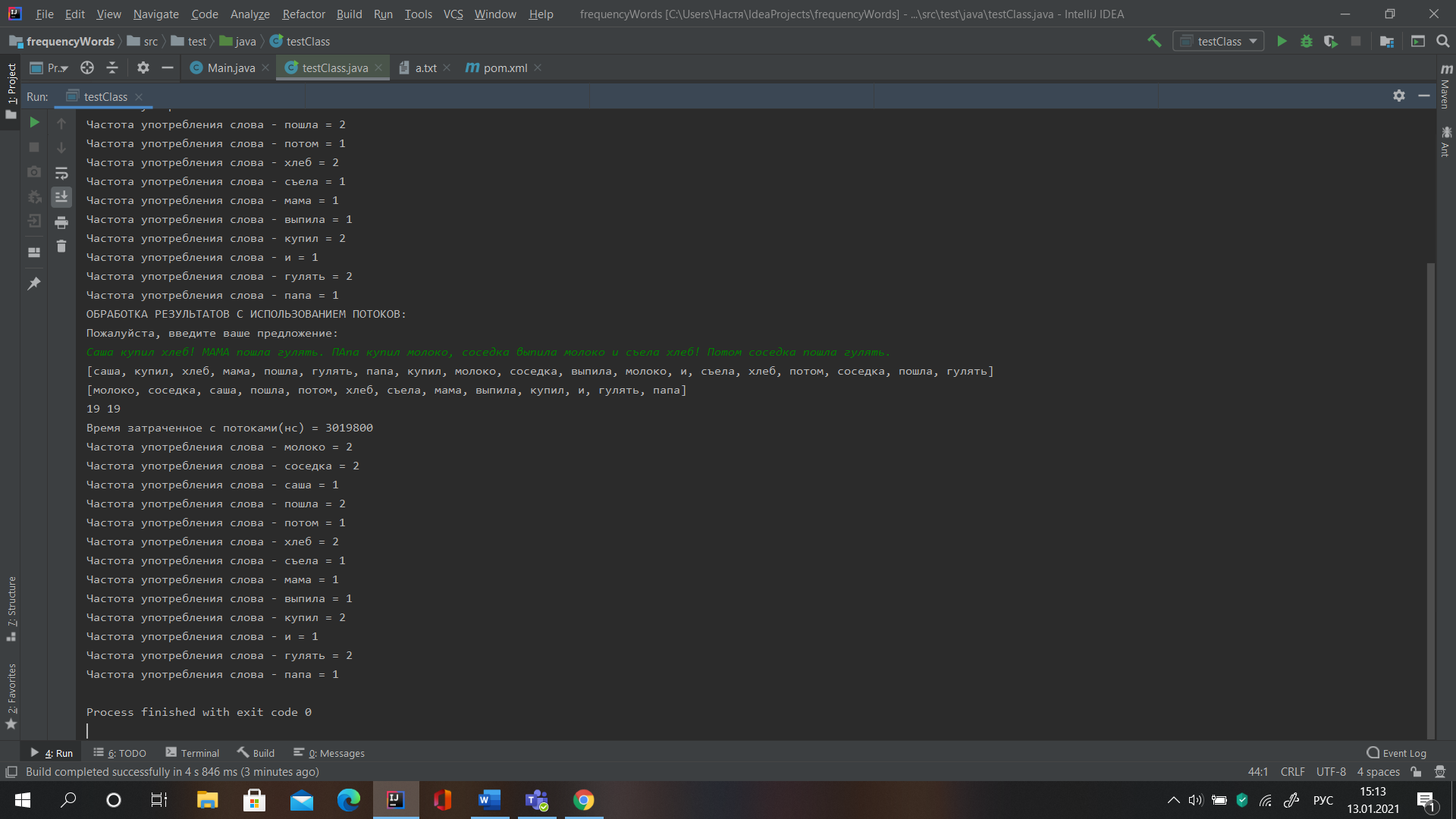
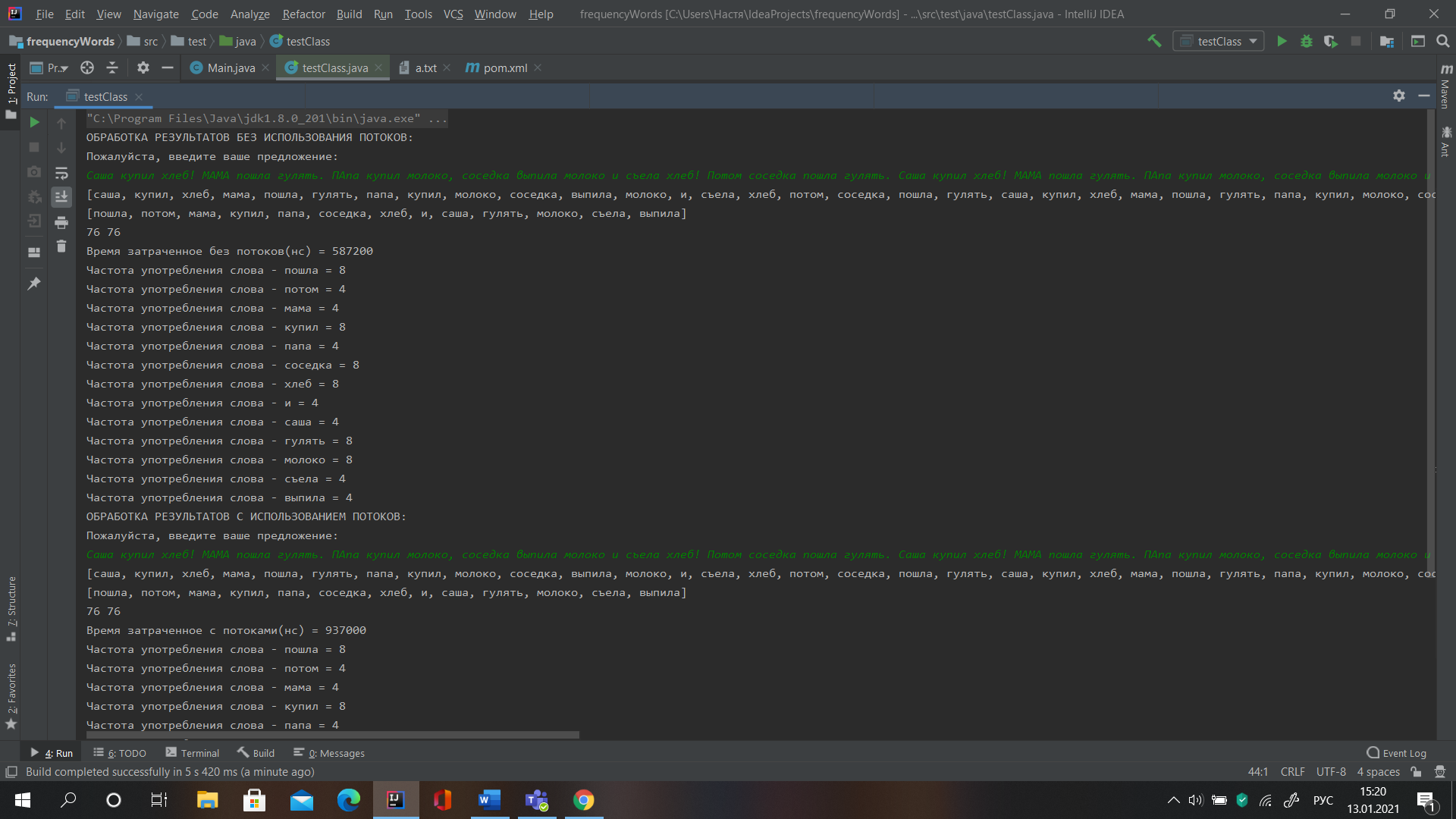


Рис.1 Результат при массиве из 19 элементов

На рисунках выше мы можем увидеть результаты работы программ как многопоточной, так и однопоточной. И как мы можем заметить, результаты совпадают, что говорит о правильности работы обеих программ.

* Теперь посмотрим, что происходит, когда мы обрабатываем массив из гораздо большего количества элементов. Например, мы обработаем ту же строку, что и выше, только в 4 раза больше. ( в результате все числа должны увеличиться в 4 раза)

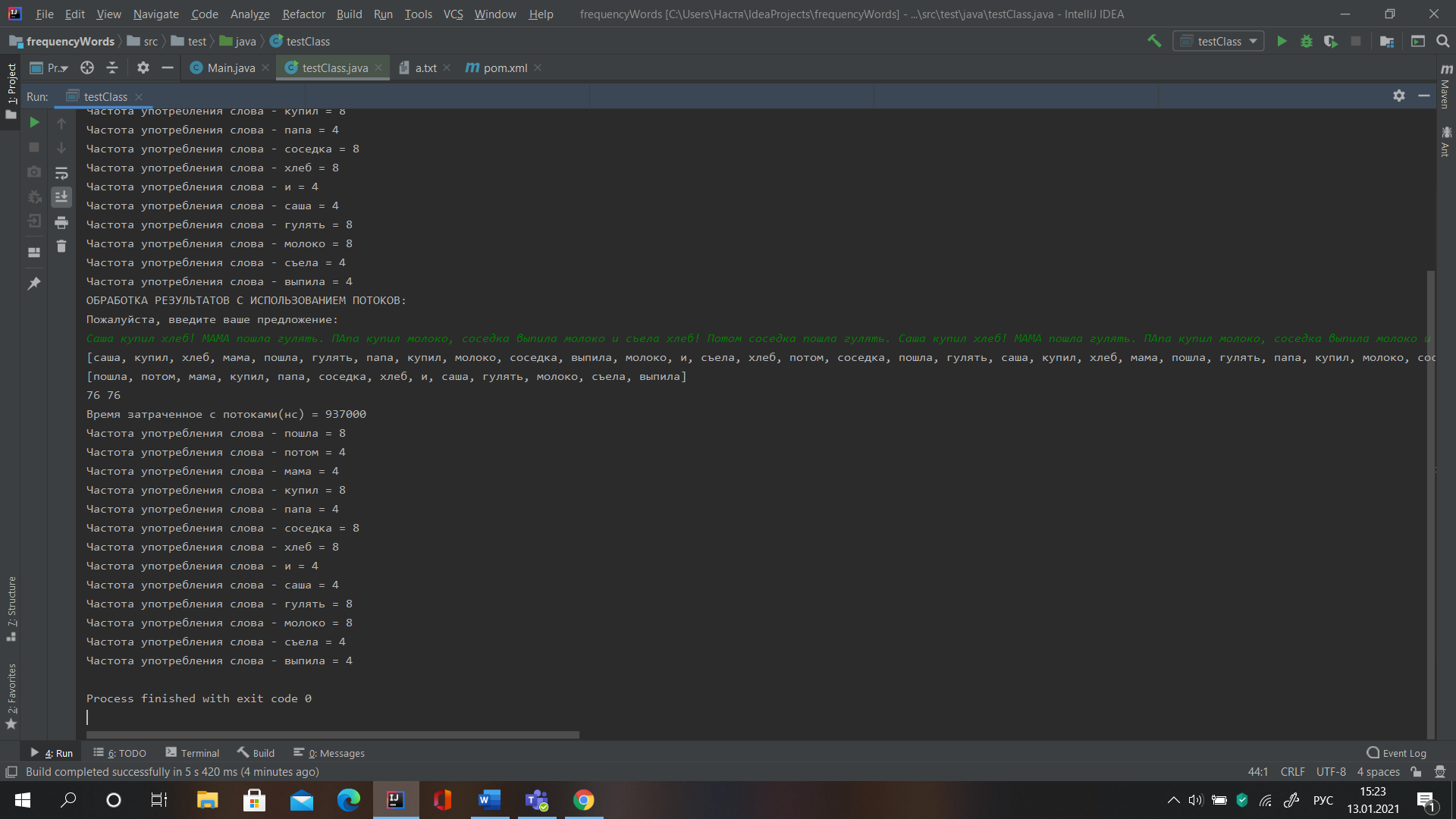
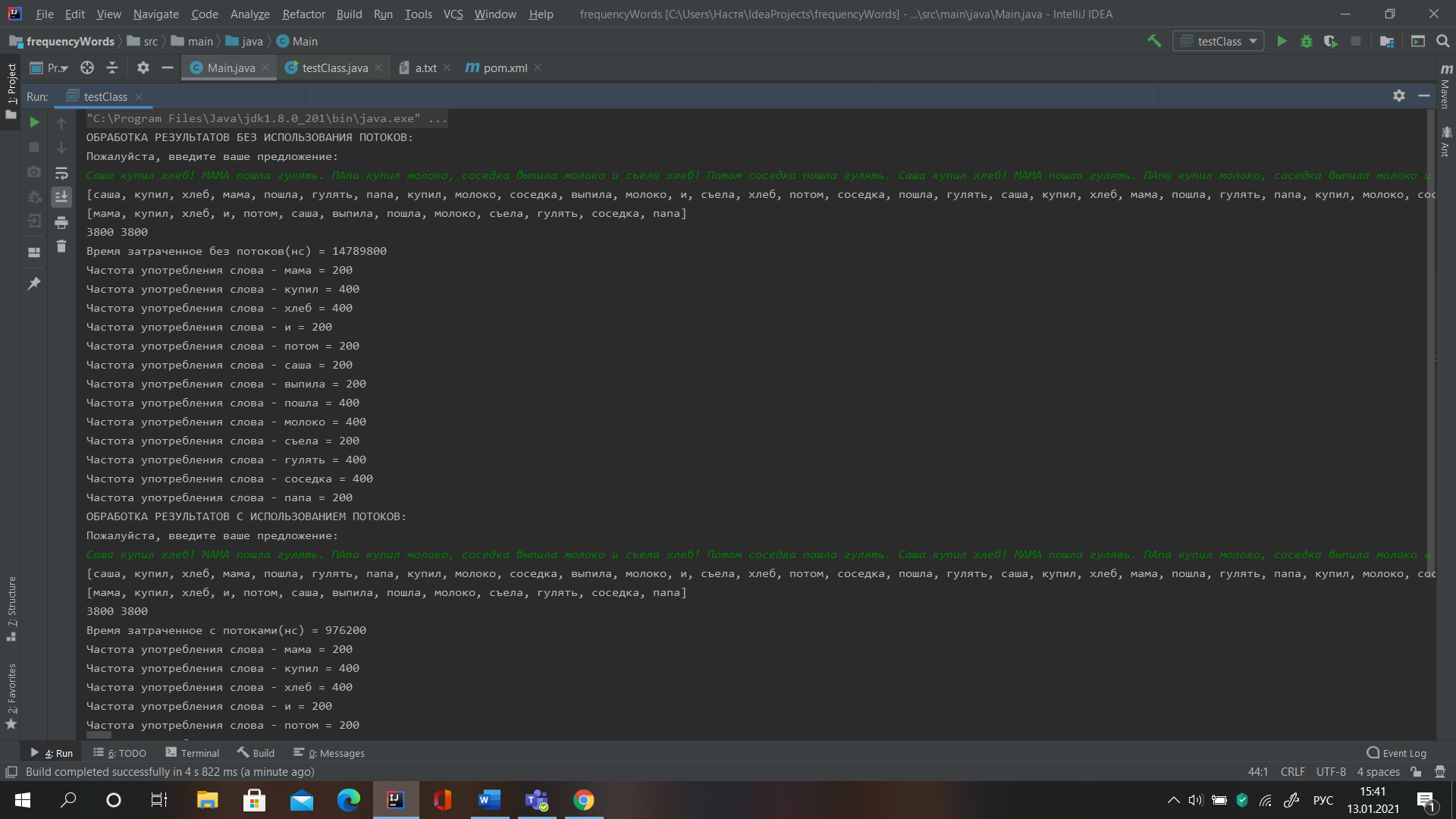
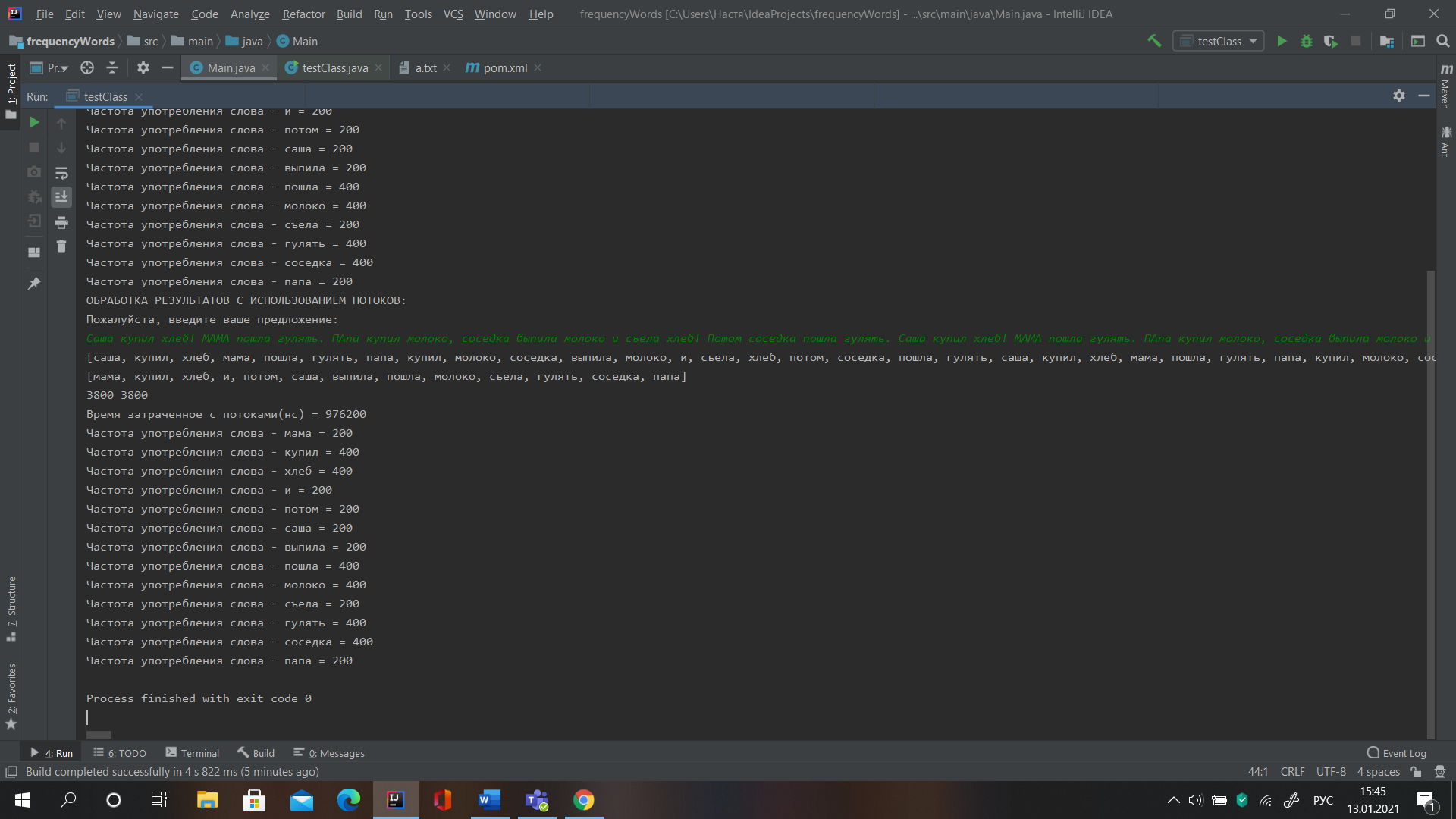


Рис.2 Результат при массиве из 76 элементов

Мы также можем пронаблюдать, что результаты в данном случае также совпадают, что говорит о корректности работы обеих программ. Наблюдая за временем, можно сказать, что однопоточная программа все еще выполняет работу быстрее, однако сравнив с прошлым временем (при 19 элементах) многопоточную программу,можно сказать, что при увеличении количества элементов, она стала работать быстрее.

Теперь еще больше увеличим размерность.

   
Рис.3: Результат при массиве из 3800 элементов

Пронаблюдав за результатами работы программ, мы также можем увидеть одинаковые результаты что говорит о корректности работ. В данном случае количество элементов уже было равно 3800 и мы можем заметить уде преимущество многопоточности перед однопоточностью с точки зрения времени.

# **Вывод:**

В ходе выполнения курсовой работы были разработаны однопоточная и многопоточная реализации программы (частота употребления слов в тексте) на языке Java. И также были проанализированы работы программ с регулированием размерности массивов.

Можно сказать, что при малом количестве элементов в массиве однопоточная реализация программы будет работать быстрее. Однако же, увеличивая количество элементов и количество потоков, мы может уже пронаблюдать преимущество многопоточности. (Скорость реализации многопоточной будет преимущественно выше однопоточной)