**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

Федеральное государственное автономное   
образовательное учреждение

Высшего образования

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**

**ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Инженерная школа информационных технологий и робототехники

Направление: «Программная инженерия»

Отчет по лабораторной работе №9 по дисциплине

**«Параллельное, распределенное программирование»**

Выполнил:

Студент группы 8К61 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Исламов Е.Р.

Принял:

Доцент ОИТ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Мокина Е.Е.

Томск 2019

**Содержание**

[Цель работы 3](#_Toc29516535)

[Ход работы 4](#_Toc29516536)

[Вывод 11](#_Toc29516537)

# Цель работы

Познакомиться с интерфейсом ExecutorService.

Реализуйте программу, которая с использованием пула в 4 потока составляет

поздравительное письмо из следующего текста:

В этот \_\_\_\_\_\_\_\_\_\_\_\_\_ и \_\_\_\_\_\_\_\_\_\_\_\_\_\_праздник, когда на \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_небе горят \_\_\_\_\_\_\_\_\_\_ звезды

и за этим \_\_\_\_\_\_\_\_\_\_\_\_\_столом в этом \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_зале собрались \_\_\_\_\_\_\_\_\_\_\_\_ дамы и не менее

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_кавалеры. Желаю присутствующим \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_чудес, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ радости,

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ счастья, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ любви. И не удивительно, если к нам придут \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дед Мороз и \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Снегурочка. Они поздравят нас с \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Новым годом, раздадут

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_подарки, и мы вместе споем \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ песню. А в 12 часов пробьют

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ куранты и на всей \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_земле воцарится \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 год.

Загадывайте \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ желания, и они обязательно сбудутся! Пусть наступающий год будет для

всех нас \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ и \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_. Пусть в новом году исполнятся \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ планы.

Мы верим, что Новый 2020 Год принесет всем собравшимся много \_\_\_\_\_\_\_\_\_\_\_\_\_\_ приключений и

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ эмоций.

Прилагательные не должны повторяться в письме, если попадается повторяющееся прилагательное выполнение действий потоком откладывается на 5 секунд.

Обработайте исключение, описанными выше методам, если прилагательные в файле

закончились (продемонстрируйте это с помощью файла с меньшим количеством

прилагательных).

# Ход работы

В ходе работы была разработана программа, позволяющая заполнить шаблон письма случайными прилагательными из файла в разных потоках.

Составление письма осуществляет класс LetterMaker. Он служит точкой старта и получения данных потоков пула потоков.

**package** Lab9;  
  
**import** java.util.ArrayList;  
**import** java.util.List;  
**import** java.util.concurrent.\*;  
**import** java.util.stream.IntStream;  
  
*/\*\*  
 \* Многопоточный составитель письма.  
 \*/***class** LetterMaker {  
 */\*\*  
 \* Число потоков.  
 \*/* **private static final int *THREADS\_COUNT*** = 4;  
  
 */\*\*  
 \* Контроллер использования слов.  
 \*/* **private** AlreadyUsedWordsController **alreadyUsedWordsController**;  
  
 */\*\*  
 \* Список слов.  
 \*/* **private** List<String> **words**;  
  
 */\*\*  
 \* Шаблон, в который нужно вставлять слова.  
 \*/* **private** String **template**;  
  
 */\*\*  
 \* Конструктор.  
 \*  
 \** ***@param template*** *Шаблон.  
 \** ***@param words*** *Слова.  
 \*/* LetterMaker(String template, List<String> words) {  
 **this**.**template** = template;  
 **this**.**words** = words;  
 **this**.**alreadyUsedWordsController** = **new** AlreadyUsedWordsController();  
 }  
  
 */\*\*  
 \* Разбить шаблон на части для разных потоков.  
 \*  
 \** ***@param template*** *Шаблон.  
 \** ***@return*** *Массив частей шаблона.  
 \*/* **private static** String[] SplitTemplate(String template) {  
 **int** length = (**int**) Math.*ceil*((**float**) template.length() / (**float**) ***THREADS\_COUNT***);  
 **return** IntStream  
 .*range*(0, (template.length() + length - 1) / length)  
 .mapToObj(i -> template.substring(i \*= length, Math.*min*(i + length, template.length())))  
 .toArray(String[]::**new**);  
 }  
  
 */\*\*  
 \* Подготовить письмо по заданному шаблону с использованием заданных слов. Слова не могут повторяться.  
 \*  
 \** ***@return*** *Письмо.  
 \*/* String MakeLetter() {  
 ExecutorService executor = Executors.*newFixedThreadPool*(***THREADS\_COUNT***);  
  
 List<Future<String>> futures = **new** ArrayList<Future<String>>();  
  
 String[] parts = *SplitTemplate*(**template**);  
 **for** (**int** i = 0; i < ***THREADS\_COUNT***; i++) {  
 Callable<String> filler = **new** LetterPartFiller(parts[i], **words**, **alreadyUsedWordsController**);  
 futures.add(executor.submit(filler));  
 }  
 executor.shutdown();  
 StringBuilder result = **new** StringBuilder();  
 **for** (Future future : futures) {  
 **try** {  
 result.append(future.get());  
 } **catch** (InterruptedException | ExecutionException e) {  
 **return** e.getMessage();  
 }  
 }  
 **return** result.toString();  
 }  
}

При формировании письма оно разбивается на части и заполнение каждой части происходит в своем потоке пула потоков. За заполнение одной части отвечает класс LetterPartFiller

**package** Lab9;  
  
**import** java.util.List;  
**import** java.util.concurrent.Callable;  
**import** java.util.concurrent.ThreadLocalRandom;  
  
*/\*\*  
 \* Наполнитель части шаблона словами.  
 \*/***class** LetterPartFiller **implements** Callable<String> {  
 */\*\*  
 \* Часть шаблона.  
 \*/* **private** String **letterPart**;  
  
 */\*\*  
 \* Слова.  
 \*/* **private** List<String> **words**;  
  
 */\*\*  
 \* Контроллер использования слов.  
 \*/* **private** AlreadyUsedWordsController **alreadyUsedWordsController**;  
  
 */\*\*  
 \* Конструктор.  
 \*  
 \** ***@param letterPart*** *Часть шаблона.  
 \** ***@param words*** *Слова.  
 \** ***@param alreadyUsedWordsController*** *Котнроллер использования слов.  
 \*/* LetterPartFiller(String letterPart, List<String> words,  
 AlreadyUsedWordsController alreadyUsedWordsController) {  
  
 **this**.**letterPart** = letterPart;  
 **this**.**words** = words;  
 **this**.**alreadyUsedWordsController** = alreadyUsedWordsController;  
 }  
  
 */\*\*  
 \* Непосредственное формирование результата.  
 \*  
 \** ***@return*** *Заполненная часть шаблона.  
 \** ***@throws*** *Exception Исключение.  
 \*/* @Override  
 **public** String call() **throws** Exception {  
 StringBuilder result = **new** StringBuilder();  
 **for** (**char** c : **letterPart**.toCharArray()) {  
 **if** (c != **'#'**) {  
 result.append(c);  
 **continue**;  
 }  
  
 String word = getRandomWord();  
 **while** (**alreadyUsedWordsController**.wasUsed(word)) {  
 **int** delta = **words**.size() - **alreadyUsedWordsController**.getUsedWordsCount();  
 **if** (delta == 0)  
 **throw new** OutOfWordsException();  
 System.***out***.printf(  
 **"%s: Слово %s уже было использовано, но осталось еще %d слов. Ищу новое слово (сон 5с).\n"**,  
 Thread.*currentThread*().getName(), word, delta);  
 Thread.*sleep*(5000);  
 word = getRandomWord();  
 }  
 System.***out***.printf(**"%s: Использую слово %s.\n"**, Thread.*currentThread*().getName(), word);  
 result.append(word);  
 }  
 **return** result.toString();  
 }  
  
 */\*\*  
 \* Получить случайное слово из списка слов.  
 \*  
 \** ***@return*** *Случайное слово.  
 \*/* **private** String getRandomWord() {  
 **int** index = ThreadLocalRandom.*current*().nextInt(**words**.size());  
 **return words**.get(index);  
 }  
}

Так как, по условию, слова не должны повторяться, создан класс AlreadyUsedWordsController, позволяющий этого избежать.

**package** Lab9;  
  
**import** java.util.ArrayList;  
**import** java.util.List;  
**import** java.util.concurrent.locks.ReadWriteLock;  
**import** java.util.concurrent.locks.ReentrantReadWriteLock;  
  
*/\*\*  
 \* Контроллер использования слов.  
 \*/***class** AlreadyUsedWordsController {  
 */\*\*  
 \* Слова, которые уже использовались.  
 \*/* **private** List<String> **used** = **new** ArrayList<>();  
  
 */\*\*  
 \* Замок, чтобы чтение и запись не осуществлялись одновременно.  
 \*/* **private** ReadWriteLock **rwLock** = **new** ReentrantReadWriteLock();  
  
 */\*\*  
 \* Проверить, использовалось ли ранее указанное слово.  
 \*  
 \** ***@param string*** *Слово.  
 \** ***@return*** *Флаг использованности.  
 \*/* **boolean** wasUsed(String string) {  
 **rwLock**.writeLock().lock();  
 **try** {  
 **if** (**used**.contains(string))  
 **return true**;  
 **used**.add(string);  
 **return false**;  
 } **finally** {  
 **rwLock**.writeLock().unlock();  
 }  
 }  
  
 */\*\*  
 \* Получить количество использованных слов.  
 \*  
 \** ***@return*** *Количество использованных слов.  
 \*/* **int** getUsedWordsCount() {  
 **try** {  
 **rwLock**.readLock().lock();  
 **return used**.size();  
 } **finally** {  
 **rwLock**.readLock().unlock();  
 }  
 }  
}

Если слова в списке закончились, выбрасывается искючение OutOfWordsException

**package** Lab9;  
  
**import** java.util.concurrent.ExecutionException;  
  
*/\*\*  
 \* Исключение, говорящее о том, что закончились слова в списке слов.  
 \*/***public class** OutOfWordsException **extends** ExecutionException {  
 */\*\*  
 \** ***@return*** *Сообщение исключения.  
 \*/* @Override  
 **public** String getMessage() {  
 **return "Произошла ошибка! Закончились слова в списке."**;  
 }  
}

С помощью класса Main осуществляется предподготовка шаблона и запуск демонстрации.

**package** Lab9;  
  
**import** java.nio.file.Files;  
**import** java.nio.file.Paths;  
**import** java.util.List;  
  
*/\*\*  
 \* Лабораторная работа №9.  
 \*/***public class** Main {  
 */\*\*  
 \* Контроллер уже использованных слов.  
 \*/* AlreadyUsedWordsController **alreadyUsedWordsController** = **new** AlreadyUsedWordsController();  
  
 */\*\*  
 \* Формирование письма с достаточным для заполнения. количеством слов.  
 \*  
 \** ***@param args*** *Параметры запуска.  
 \** ***@throws*** *Exception Неожиданное исключение.  
 \*/* **public static void** main(String[] args) **throws** Exception {  
 *demo*(**"src/Lab9/Data/words.txt"**);  
 }  
  
 */\*\*  
 \* Формирование письма.  
 \*  
 \** ***@param pathToWords*** *Путь к словам для заполнения..  
 \** ***@throws*** *Exception Неожиданное исключение.  
 \*/* **public static void** demo(String pathToWords) **throws** Exception {  
 String template = *prepareBaseTemplate*();  
 List<String> words = Files.*readAllLines*(Paths.*get*(pathToWords));  
  
 LetterMaker letterMaker = **new** LetterMaker(template, words);  
  
 System.***out***.println(**"\n\nРезультат:\n\n"** + letterMaker.MakeLetter());  
 }  
  
 */\*\*  
 \* Подготовить шаблон (поменять \_\_\_\_\_ на #) из исходного.  
 \*  
 \** ***@return*** *Шаблон. На места % нужно вставить слова.  
 \** ***@throws*** *Exception Неожиданное исключение.  
 \*/* **private static** String prepareBaseTemplate() **throws** Exception {  
 String template = **new** String(Files.*readAllBytes*(Paths.*get*(**"src/Lab9/Data/input.txt"**)));  
  
 StringBuilder s = **new** StringBuilder();  
 **for** (**int** i = 0, n = template.length(); i < n; i++) {  
 **char** c = template.charAt(i);  
 **if** (c == **'\_'**) {  
 s.append(**"#"**);  
 **while** (template.charAt(i) == **'\_'**) {  
 i++;  
 }  
 i--;  
 } **else** {  
 **if** (s.length() > 0 && c != **' '** && s.charAt(s.length() - 1) == **'#'**)  
 s.append(**" "**);  
 s.append(c);  
 }  
 }  
 **return** s.toString();  
 }  
}

Формирование письма с недостаточным для заполнения слов (ошибкой) производится с помощью класса DemoOutOfWords

**package** Lab9;  
  
*/\*\*  
 \* Неудачный запуск.  
 \*/***public class** DemoOutOfWords {  
 */\*\*  
 \* Формирование письма с недостаточным для заполнения. количеством слов.  
 \*  
 \** ***@param args*** *Параметры запуска.  
 \** ***@throws*** *Exception Неожиданное исключение.  
 \*/* **public static void** main(String[] args) **throws** Exception {  
 Main.*demo*(**"src/Lab9/Data/wordsreduced.txt"**);  
 }  
}

На рисунках 1 и 2 приведен результат удачной и неудачной работы программы.

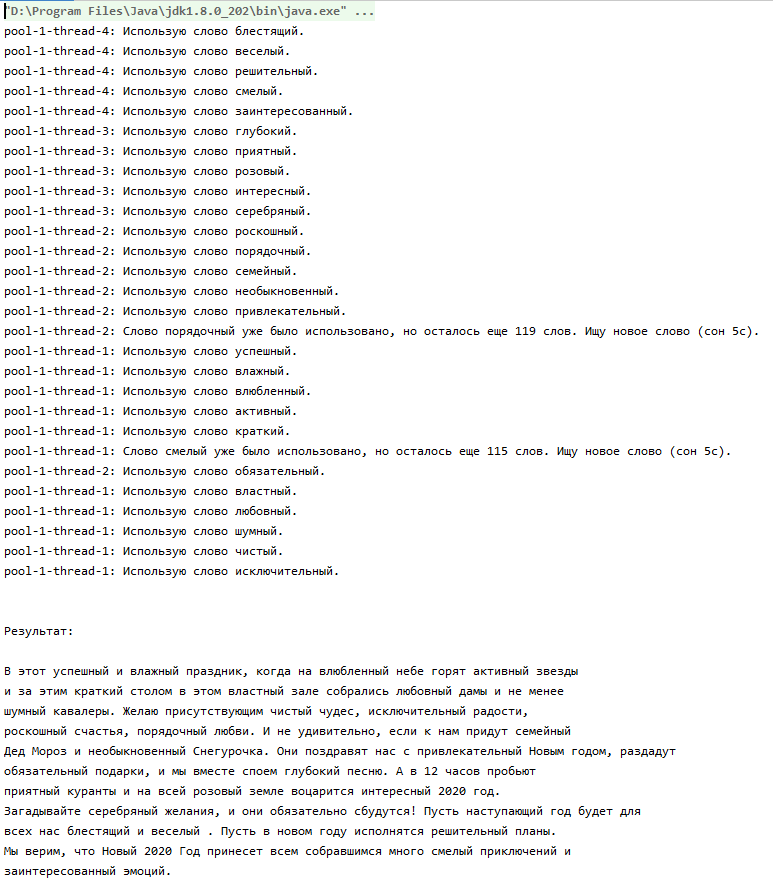


Рисунок 1 - результат удачной работы программы

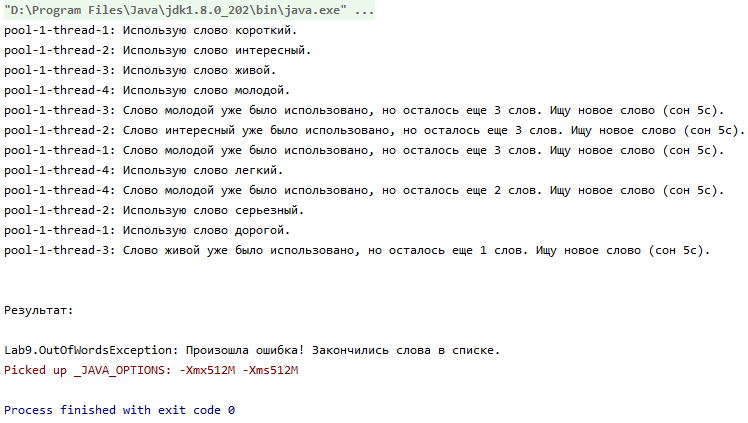


Рисунок 2 – неудачная работа программы

# Вывод

В результате выполненной работы были получены навыки по работе с пулом потоков на примере формирования поздравительного письма.