**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 №3 по дисциплине

**«Параллельное, распределенное программирование»**

Выполнил:

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

Принял:

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

Томск 2019

**Содержание**

[Цель работы 3](#_Toc23454460)

[Ход работы 4](#_Toc23454461)

[Вывод 6](#_Toc23454462)

# Цель работы

Познакомиться с синхронизацией потоков в Java.

# Ход работы

В ходе работы был реализован класс Sync, считывающий стихотворение из файла, а потом выводящий его с использованием многопоточности.

**public class** Sync {  
 */\*\*  
 \* Вывод стихотворения в многопоточном режиме.  
 \*/* **public static void** main(String[] args) **throws** Exception{  
 *// Считываем стихотворение из файла.* File file = **new** File(**"src/Lab3/Demo/input.txt"**);  
 BufferedReader br = **new** BufferedReader(**new** FileReader(file));  
 List<String> strings = **new** ArrayList<String>();  
 String st;  
 **while** ((st = br.readLine()) != **null**)  
 strings.add(st);  
  
 *// Создаем потоки для каждой строки.* CallMe target = **new** CallMe();  
 List<Caller> callers = strings  
 .stream()  
 .map(x -> **new** Caller(target, x))  
 .collect(Collectors.*toList*());  
  
 *// Ожидаем завершение потока.* **for** (Caller caller : callers) {  
 caller.**t**.join();  
 }  
 }  
}

Непосредственный вывод производится в классе CallMe:

**public class** CallMe {  
 */\*\*  
 \* Вывод сообщения.  
 \** ***@param msg*** *Сообщение.  
 \*/* **void** call(String msg) {  
 System.***out***.print(**"["** + msg);  
 **try** {  
 Thread.*sleep*(1000);  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 System.***out***.println(**"]"**);  
 }  
}

Вызов метода CallMe.call производится с помощью класса Caller:

**public class** Caller **implements** Runnable {  
 **public** Thread **t**;  
 String **message**;  
 CallMe **target**;  
  
 **public** Caller(CallMe target, String message) {  
 **this**.**target** = target;  
 **this**.**message** = message;  
 **t** = **new** Thread(**this**);  
 **t**.start();  
 }  
  
 */\*\*  
 \* Вызывает метод call цели без синхронизации.  
 \*/* **public void** run() {  
 **target**.call(**message**);  
 }  
}

Также в рамках работы реализован класс SynchronizedCaller, вызывающий метод CallMe.call с использованием синхронизации:

**public class** SynchronizedCaller **extends** Caller {  
 **public** SynchronizedCaller(CallMe target, String message) {  
 **super**(target, message);  
 }  
  
 */\*\*  
 \* Вызывает метод call цели с синхронизацией.  
 \*/* @Override  
 **public void** run() {  
 **synchronized** (**target**) {  
 **target**.call(**message**);  
 }  
 }  
}

На рисунке 1 приведен результат работы программы без использования синхронизации потоков.

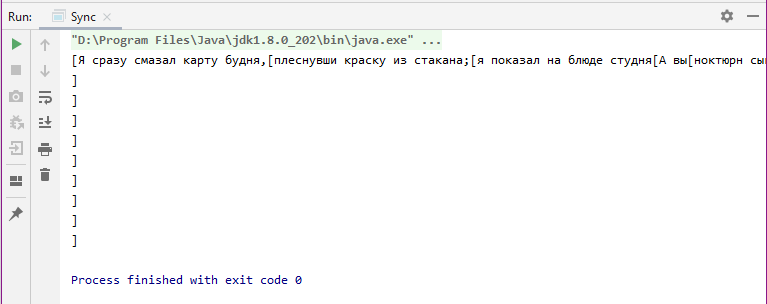


Рисунок 1 - результат работы программы

На рисунке 2 приведен результат работы программы с использованием синхронизации потоков.

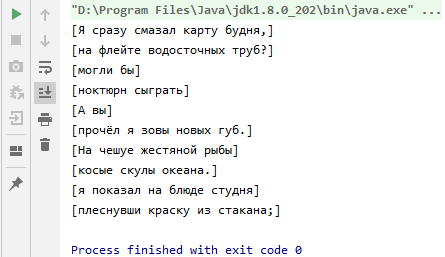


Рисунок 2 - результат работы программы

Из рисунка 2 видно, что строки выводятся правильно. Однако порядок вывода строк неверный. Это связано с тем, что синхронизация потоков с использованием synchronized не гарантирует правильность порядка вывода.

# Вывод

В результате выполненной работы были улучшены навыки по работе с потоками Java, а также способы синхронизации потоков.