**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 №5 по дисциплине

**«Параллельное, распределенное программирование»**

Выполнил:

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

Принял:

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

Томск 2019

**Содержание**

[Цель работы 3](#_Toc23456838)

[Задание 1 4](#_Toc23456839)

[Задание 2 5](#_Toc23456840)

[Вывод 7](#_Toc23456841)

# Цель работы

Познакомиться с взаимной блокировкой, а также с приостановкой, возобновлением и остановкой потоков исполнения.

# Задание 1

В рамках задачи 1 реализованы классы, позволяющие добиться взаимной блокировки. В двух параллельных потоках блокируются объекты a и b, после чего эти заблокированные объекты пытаются обратиться к друг другу, приводя к дедлоку.

*/\*\*  
 \* Взаимная блокировка.  
 \*/***public class** Deadlock **implements** Runnable {  
 A **a** = **new** A();  
 B **b** = **new** B();  
  
 Deadlock() {  
 Thread.*currentThread*().setName(**"Главный поток"**);  
 Thread t = **new** Thread(**this**, **"Соперничающий поток"**);  
 t.start();  
  
 *// Блокируем a из главного потока.* **a**.foo(**b**);  
 System.***out***.println(**"Назад в главный поток"**);  
 }  
  
 **public void** run() {  
 *// Блокируем b другого потока.* **b**.bar(**a**);  
 System.***out***.println(**"Назад в другой поток"**);  
 }  
  
 **public static void** main(String[] args) {  
 **new** Deadlock();  
 }  
}

**class** A {  
 **synchronized void** foo(B b) {  
 String name = Thread.*currentThread*().getName();  
 System.***out***.println(name + **" вошел в метод A.foo()"**);  
  
 **try** {  
 Thread.*sleep*(1000);  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 System.***out***.println(name + **"пытается вызвать метод B.last()"**);  
 *// Вызов метода заблокированного объекта.* b.last();  
 }  
  
 **synchronized void** last() {  
 System.***out***.println(**"В методе A.last()"**);  
 }  
}

**class** B {  
 **synchronized void** bar(A a) {  
 String name = Thread.*currentThread*().getName();  
 System.***out***.println(name + **" вошел в метод B.foo()"**);  
  
 **try** {  
 Thread.*sleep*(1000);  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 System.***out***.println(name + **"пытается вызвать метод a.last()"**);  
 *// Вызов метода заблокированного объекта.* a.last();  
 }  
  
 **synchronized void** last() {  
 System.***out***.println(**"В методе B.last()"**);  
 }  
}

На рисунке 1 приведен результат работы программы (программа никогда не завершится).

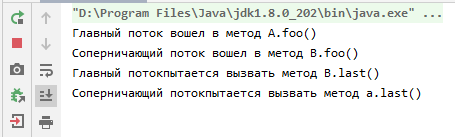


Рисунок 1 - результат работы программы

# Задание 2

В рамках задачи 2 реализован класс MyThread, считывающий стихотворение из файла и выводящий его в консоль. Класс поддерживает приостановку и возобновление вывода.

**public class** MyThread **implements** Runnable {  
 */\*\*  
 \* Наименование потока.  
 \*/* **private final** String **name**;  
 */\*\*  
 \* Флаг ожидания.  
 \*/* **private boolean suspendFlag**;  
  
 */\*\*  
 \* Конструктор.  
 \*  
 \** ***@param name*** *Наименование потока.  
 \*/* MyThread(String name) {  
 **this**.**name** = name;  
 Thread thread = **new** Thread(**this**, name);  
 System.***out***.println(**"Новый поток: "** + thread);  
 **suspendFlag** = **false**;  
 thread.start();  
 }  
  
 */\*\*  
 \* Тело потока.  
 \*/* @Override  
 **public void** run() {  
 **try** {  
 File file = **new** File(**"src/Lab3/Demo/input.txt"**);  
 BufferedReader br = **new** BufferedReader(**new** FileReader(file));  
 String st;  
 **while** ((st = br.readLine()) != **null**) {  
 System.***out***.println(**name** + **": "** + st);  
 Thread.*sleep*(1000);  
 **synchronized** (**this**) {  
 **while** (**suspendFlag**) {  
 wait();  
 }  
 }  
 }  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Приостановить выполнение потока.  
 \*/* **synchronized void** suspend() {  
 **suspendFlag** = **true**;  
 }  
  
 */\*\*  
 \* Возобновить выполнение потока.  
 \*/* **synchronized void** resume() {  
 **suspendFlag** = **false**;  
 notify();  
 }  
}

С помощью класса Demo производится создание потоков, а также приостановка и возобновление потоков.

**public class** Demo {  
 **public static void** main(String[] args) **throws** Exception{  
 MyThread t1 = **new** MyThread(**"Первый поток"**);  
 MyThread t2 = **new** MyThread(**"Второй поток"**);  
  
 Thread.*sleep*(1500);  
 t1.suspend();  
 Thread.*sleep*(4000);  
 t1.resume();  
 t2.suspend();  
 Thread.*sleep*(4000);  
 t2.resume();  
 }  
}

На рисунке 1 приведен результат работы программы (программа никогда не завершится).

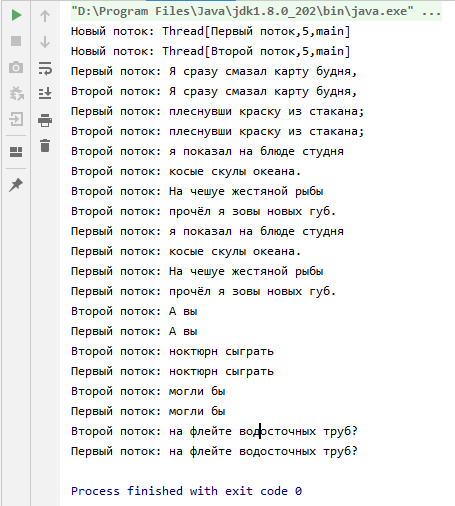


Рисунок 1 - результат работы программы

Как следует из рисунка, в какие-то промежутки времени выполняются оба потока, а в другие – только один из потоков, что соответствует требованиям задания.

# Вывод

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