

Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и кибербезопасности
Высшая школа программной инженерии

Курсовая работа

по дисциплине «Объектно-ориентированное программирование»

на тему

«Супервизор»

Выполнил: Симоненко Иван Сергеевич

Группа: 5130904/20002

Преподаватель: Эйзенах Д. С.

Санкт-Петербург

2024

Оглавление

1	Постановка задачи	3
2	Описание выполненных работ	3
3	Диаграмма классов	4
4	Исходный код программы	4
5	Результат работы программы.....	7
6	Вывод	7

1 Постановка задачи

Создать супервизор (управляющую программу), которая контролирует исполнение абстрактной программы.

Абстрактная программа работает в отдельном потоке и является классом с полем перечисляемого типа, который отражает ее состояние

- UNKNOWN – перед первым запуском
- STOPPING - остановлена
- RUNNING - работает
- FATAL ERROR – критическая ошибка

и имеет поток-демон случайного состояния, который в заданном интервале меняет её состояние на случайное.

У супервизора должны быть методы остановки и запуска абстрактной программы, которые меняют ее состояние. Супервизор является потоком, который циклически опрашивает абстрактную программу, и если ее состояние STOPPING, то перезапускает ее. Если состояние FATAL ERROR, то работа абстрактной программы завершается супервизором. Все изменения состояний должны сопровождаться соответствующими сообщениями в консоли. Супервизор не должен пропустить ни одного статуса абстрактной программы. Использовать конструкции с wait/notify.

2 Описание выполненных работ

Разработка классов и их взаимодействие:

- Реализованы классы Process, AbstractProgram, Supervisor, и Utils.
- Определены перечисления (enum) для состояния программы (State).

Реализация многопоточности:

- В классе AbstractProgram создаётся и запускается демон-поток для изменения состояния программы.
- В классе Supervisor поток супервизора отслеживает состояние программы и принимает решения о перезапуске или остановке.

Синхронизация потоков:

- Использованы блоки synchronized и методы wait/notify для координации работы между потоками.
- Обеспечена корректная видимость изменений состояния программы между потоками.

3 Диаграмма классов

Класс	Атрибуты	Методы
Process	- state: State	+ main(String[] args): void
	- mutex: Object	
	- abstractProgram: Thread	
AbstractProgram		+ run(): void
		- someWork(): void
Supervisor		+ run(): void
		- runProgram(): void
		- stopProgram(): void
Utils		+ pause(int min, int max): void
State		+ RUNNING
		+ FATAL_ERROR
		+ UNKNOWN
		+ STOPPING

4 Исходный код программы

[Process.java](#)

```
package lab;

import java.util.Random;

public class Process {

    private static State state = State.UNKNOWN;
    private static final Object mutex = new Object();
    private static final Thread abstractProgram = new Thread(new AbstractProgram());

    static class AbstractProgram implements Runnable {
        @Override
        public void run() {
            System.out.println("Демон$~ Состояние демона перед запуском программы: " + state + ".");
            Thread daemon = new Thread(() -> {
                while (true) {
                    Utils.pause(500, 5500);
                    if (abstractProgram.isInterrupted()) {
                        break;
                    }
                    synchronized (mutex) {
                        do {
                            state = State.values()[new Random().nextInt(State.values().length)];
                        } while (state == State.UNKNOWN);
                        if (state.equals(State.RUNNING)) {

```

```

        System.out.println("Демон$~ Программе повезло и со-
стояние осталось. Состояние: RUNNING.");
    } else {
        System.out.println("Демон$~ Я изменил состояние про-
граммы на: " + state + ".");
    }
    mutex.notify();
}

});
daemon.setDaemon(true);
daemon.start();
System.out.println("Абстрактная программа$~ я заработала и запустила
демона!");

while (!Thread.currentThread().isInterrupted()) {
    someWork();
}

private void someWork() {
    int amogus = 0;
    amogus++;
}

static class Supervisor implements Runnable {
    @Override
    public void run() {
        System.out.println("Супервизор$~ я встал!");
        abstractProgram.start();
        while (!abstractProgram.isInterrupted()) {
            synchronized (mutex) {
                try {
                    mutex.wait();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                switch (state) {
                    case FATAL_ERROR -> stopProgram();
                    case STOPPING -> runProgram();
                    default -> System.out.println("Супервизор$~ Я ничего не
сделал.");
                }
            }
        }
    }

    private void runProgram() {
        state = State.RUNNING;
        System.out.println("Супервизор$~ Я перезапустил программу.");
    }

    private void stopProgram() {
        abstractProgram.interrupt();
        System.out.println("Супервизор$~ Я остановил программу.");
    }

    public static void main(String[] args) {
        new Thread(new Supervisor()).start();
    }
}

```

[State.java](#)

```
package lab;

public enum State {
    UNKNOWN,
    STOPPING,
    RUNNING,
    FATAL_ERROR
}
```

[Utils.java](#)

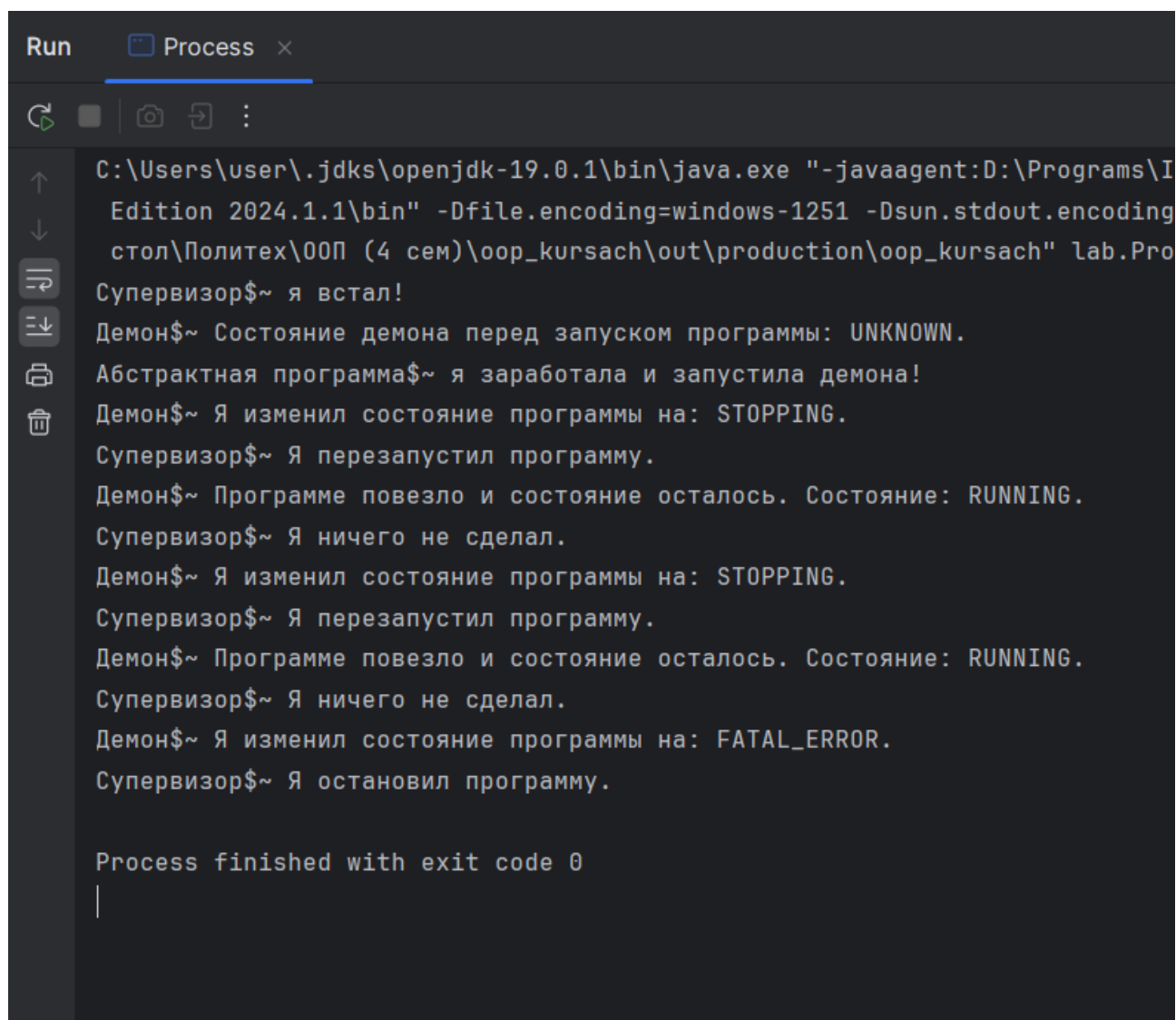
```
package lab;

import java.util.Random;

public class Utils {

    public static void pause(int lowerBound, int upperBound) {
        try {
            Thread.sleep(new Random().nextInt(upperBound - lowerBound) + lower-
Bound);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

5 Результат работы программы



```
Run Process x
C:\Users\user\.jdk\openjdk-19.0.1\bin\java.exe "-javaagent:D:\Programs\I
Edition 2024.1.1\bin" -Dfile.encoding=windows-1251 -Dsun.stdout.encoding
стол\Политех\00П (4 сем)\oop_kursach\out\production\oop_kursach" lab.Pro
Супервизор$~ я встал!
Демон$~ Состояние демона перед запуском программы: UNKNOWN.
Абстрактная программа$~ я заработала и запустила демона!
Демон$~ Я изменил состояние программы на: STOPPING.
Супервизор$~ Я перезапустил программу.
Демон$~ Программе повезло и состояние осталось. Состояние: RUNNING.
Супервизор$~ Я ничего не сделал.
Демон$~ Я изменил состояние программы на: STOPPING.
Супервизор$~ Я перезапустил программу.
Демон$~ Программе повезло и состояние осталось. Состояние: RUNNING.
Супервизор$~ Я ничего не сделал.
Демон$~ Я изменил состояние программы на: FATAL_ERROR.
Супервизор$~ Я остановил программу.

Process finished with exit code 0
|
```

6 Вывод

Разработка данной многопоточной программы позволила на практике применить и закрепить знания по темам многопоточности, синхронизации и управления состоянием в Java. Программа иллюстрирует важность правильного использования механизмов синхронизации для обеспечения корректного взаимодействия потоков. В результате работы создана система, способная адаптироваться к изменяющимся условиям и корректно реагировать на различные состояния. Полученные знания и опыт могут быть применены в более сложных системах и проектах, требующих надежного многопоточного взаимодействия.