МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Национальный исследовательский университет ИТМО»

##### ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

## **ЛАБОРАТОРНАЯ РАБОТА №4**

### по дисциплине

### «ОСНОВЫ ПРОГРАММНОЙ ИНЖЕНЕРИИ»

### Вариант № 321901

***Выполнили:***

Студенты группы P3219

Билобрам Денис,

Таранов Кирилл

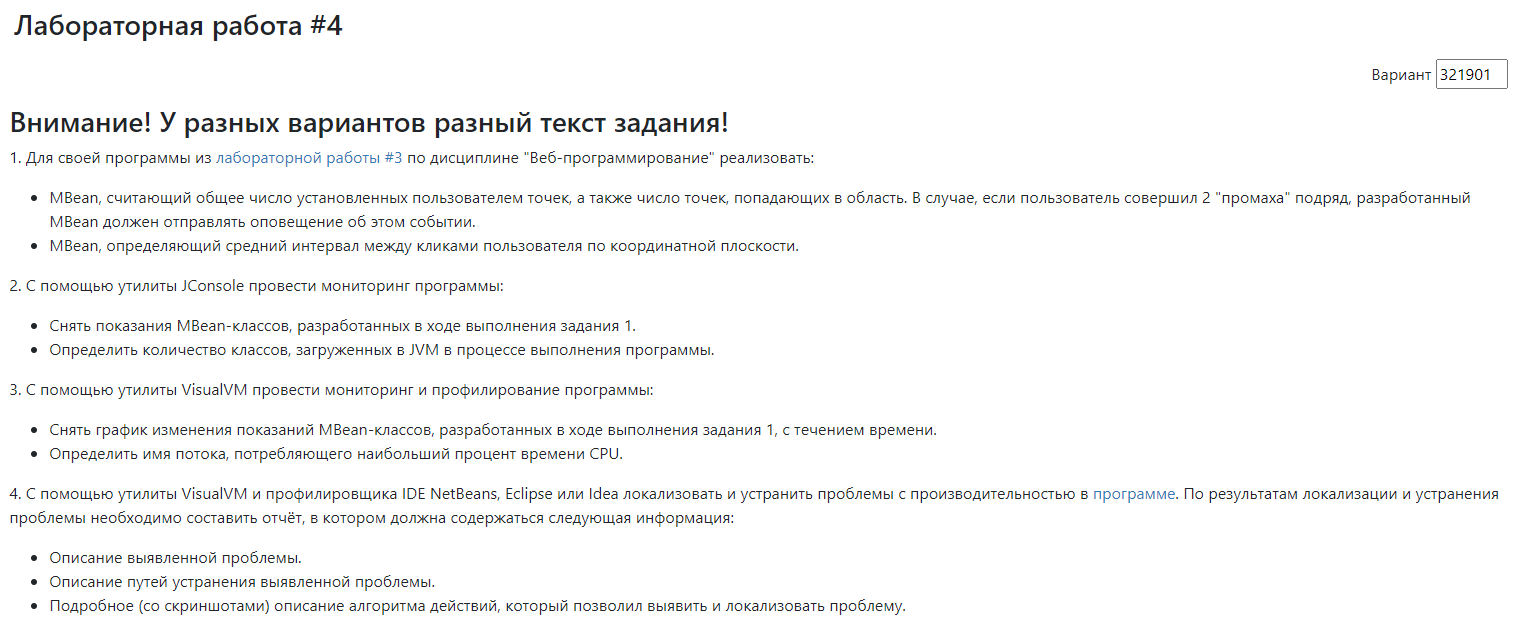
#### *Преподаватель:*

Наумова Надежда

Александровна

Санкт-Петербург, 2024

**1. Задание**



**2. Код реализованных MBean**

public interface RequestsMonitorMBean {

    int getTotalPoints();

    int getPointsInArea();

    void clearMonitor();

}

public class RequestsMonitor extends NotificationBroadcasterSupport implements RequestsMonitorMBean {

    private int totalPoints = 0;

    private int pointsInArea = 0;

    private int consecutiveMisses = 0;

    private long sequenceNumber = 1;

    @Override

    public int getTotalPoints() {

        return totalPoints;

    }

    @Override

    public int getPointsInArea() {

        return pointsInArea;

    }

    @Override

    public void clearMonitor() {

        totalPoints = 0;

        pointsInArea = 0;

        consecutiveMisses = 0;

        sequenceNumber = 0;

    }

    public void addPoint(boolean inArea) {

        totalPoints++;

        if (inArea) {

            pointsInArea++;

            consecutiveMisses = 0;

        } else {

            consecutiveMisses++;

            if (consecutiveMisses >= 2) {

                Notification notification = new Notification(

                    "lab3test.beans.RequestsMonitor.twoConsecutiveMisses",

                    this,

                    sequenceNumber++,

                    System.currentTimeMillis(),

                    "User made two consecutive misses"

                );

                consecutiveMisses = 0;

                sendNotification(notification);

            }

        }

    }

}

public interface ClickIntervalMonitorMBean {

    double getAverageInterval();

    void clearMonitor();

}

public class ClickIntervalMonitor extends NotificationBroadcasterSupport implements ClickIntervalMonitorMBean {

    private List<Long> clickTimestamps = new ArrayList<>();

    @Override

    public double getAverageInterval() {

        if (clickTimestamps.size() < 2) {

            return 0.0;

        }

        long totalInterval = 0;

        for (int i = 1; i < clickTimestamps.size(); i++) {

            totalInterval += clickTimestamps.get(i) - clickTimestamps.get(i - 1);

        }

        return (double) totalInterval / (clickTimestamps.size() - 1);

    }

    @Override

    public void clearMonitor() {

        clickTimestamps.clear();

    }

    public void addClickTimestamp(long timestamp) {

        clickTimestamps.add(timestamp);

    }

}

public class RequestsNotificationListener implements NotificationListener {

    @Override

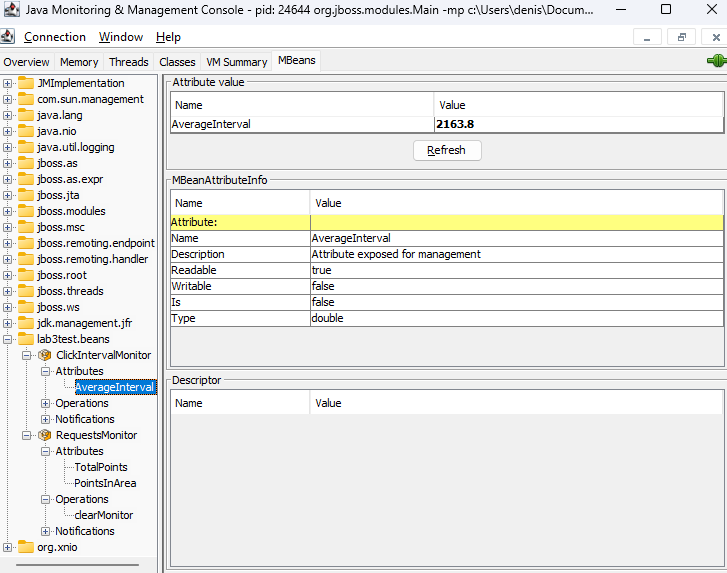
    public void handleNotification(Notification notification, Object handback) {

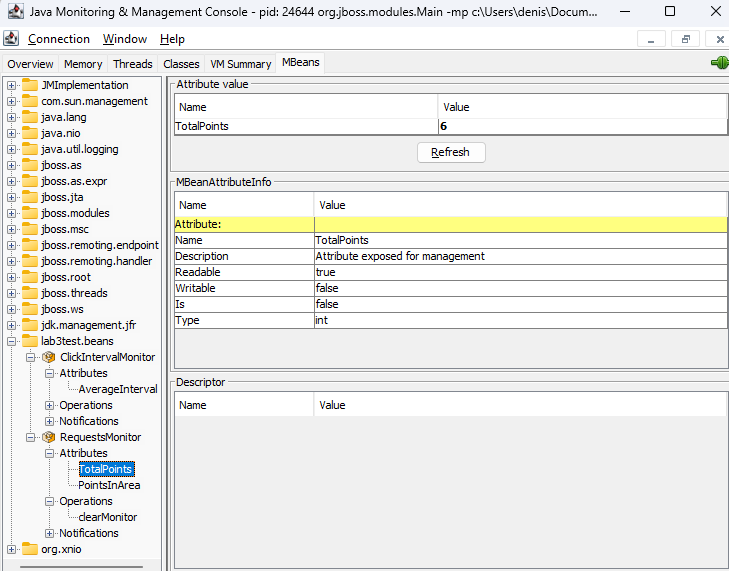
        System.out.println("Received notification: " + notification.getMessage());

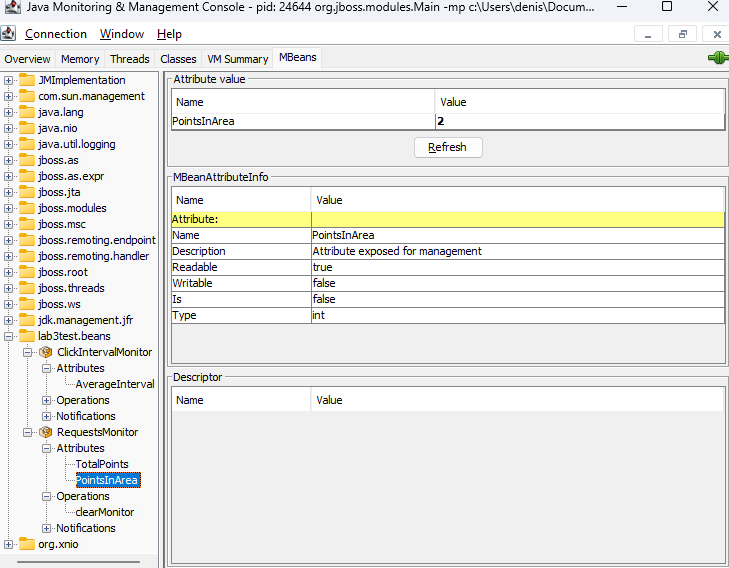
    }

}

**3. Показания MBeans с Jconsole**



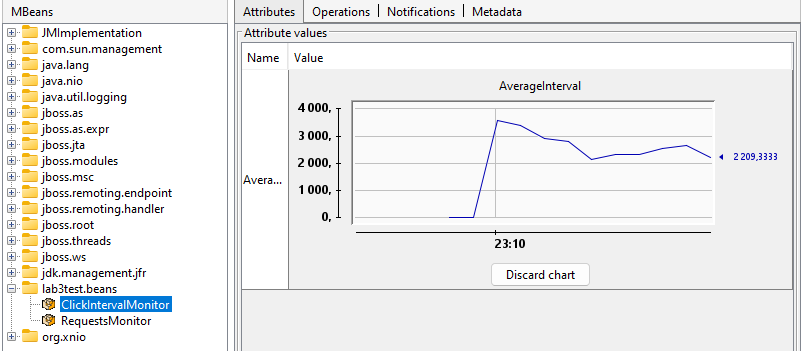


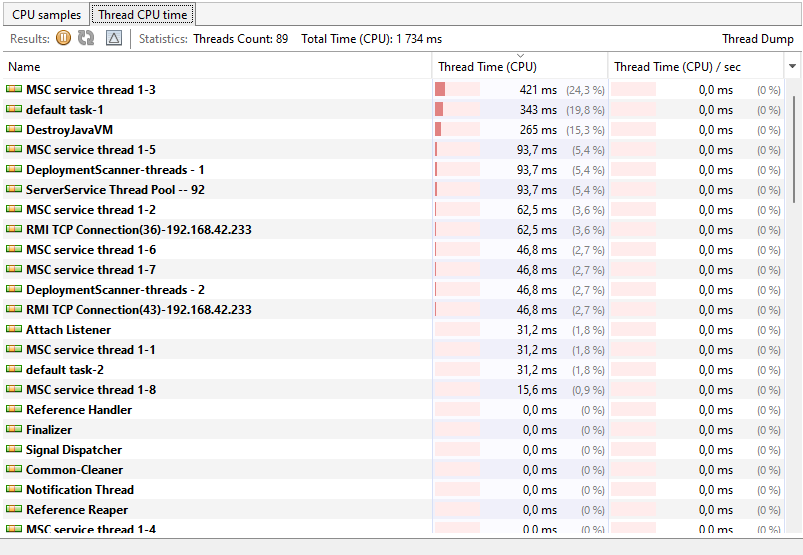




**4. Мониторинг через VisualVM**







5. Выявление проблем и устранение проблем с производительностью



Можно заметить, что CPU большую часть времени загружен на 0-1% а main находится в состоянии Sleeping. Это происходит из-за вызова в основном цикле программы:

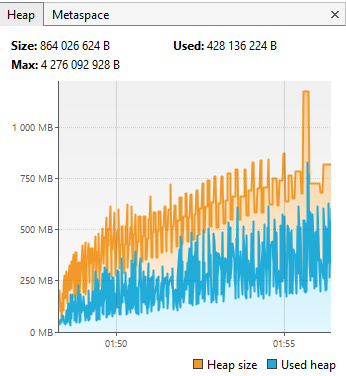
java.lang.Thread.*sleep*(200);

Для проверки на проблемы производительности и на утечки памяти поставим время сна 0 миллисекунд, теперь main будет большую часть времени находиться в Running, а CPU загружен на ~5%.

java.lang.Thread.*sleep*(0);



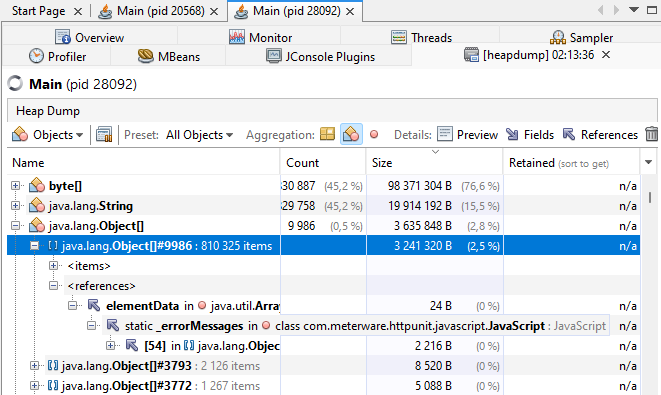
Запустим программу и проверим состояние heap. Рассмотрим, как изменяется размер используемой памяти (Used heap) и размер кучи (Heap size) со временем.



Важно обратить внимание на общий тренд используемой памяти. В данном случае видно, что после каждой сборки мусора (GC) размер используемой кучи не возвращается к исходным значениям, а имеет тенденцию к росту.

Если используется память (Used heap) постепенно увеличивается даже после нескольких сборок мусора, это может свидетельствовать о том, что происходит накопление объектов, которые не освобождаются сборщиком мусора. Это является признаком возможной утечки памяти.

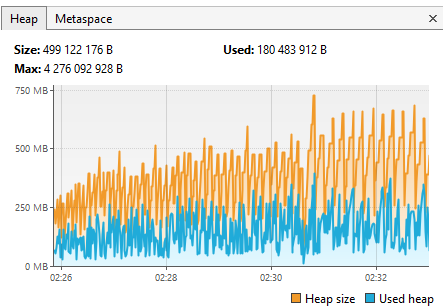
Сделаем Heap dump и посмотрим самый большой объект в heap, это массив errorMessages содержащий 810к элементов.



Находим в коде этот объект и замечаем, что происходит добавление объектов в этот ArrayList, но не происходит очищения этого списка. Из-за этого объекты скапливаются в heap, что является утечкой памяти. Найдем метод для очистки ошибок и добавим его в основной цикл программы.

while (true) {  
 WebResponse response = sc.getResponse(request);  
 System.*out*.println("Count: " + number++ + response);  
 java.lang.Thread.*sleep*(0);  
 HttpUnitOptions.*clearScriptErrorMessages*();  
}

Теперь запустим программу и проверим состояние heap:



Утечек памяти не замечено, GC работает в штатном режиме.