Федеральное государственное автономное образовательное учреждение высшего образования Национальный исследовательский университет ИТМО

Факультет программной инженерии и компьютерной техники

Основы программной инженерии

Лабораторная работа №4

Вариант № 1583

Выполнили: студенты группы P3208, Васильев Н. А., Петров В. М.

Преподаватель: Воронина Д. С.

Санкт-Петербург 2025

Текст задания

1. Для своей программы из лабораторной работы #3 по дисциплине "Веб-программирование" реализовать:

* MBean, считающий общее число установленных пользователем точек, а также число точек, попадающих в область. В случае, если количество установленных пользователем точек стало кратно 10, разработанный MBean должен отправлять оповещение об этом событии.
* MBean, определяющий площадь получившейся фигуры.

2. С помощью утилиты JConsole провести мониторинг программы:

* Снять показания MBean-классов, разработанных в ходе выполнения задания 1.
* Определить количество классов, загруженных в JVM в процессе выполнения программы.

3. С помощью утилиты VisualVM провести мониторинг и профилирование программы:

* Снять график изменения показаний MBean-классов, разработанных в ходе выполнения задания 1, с течением времени.
* Определить имя класса, объекты которого занимают наибольший объём памяти JVM; определить пользовательский класс, в экземплярах которого находятся эти объекты.

4. С помощью утилиты VisualVM и профилировщика IDE NetBeans, Eclipse или Idea локализовать и устранить проблемы с производительностью в программе. По результатам локализации и устранения проблемы необходимо составить отчёт, в котором должна содержаться следующая информация:

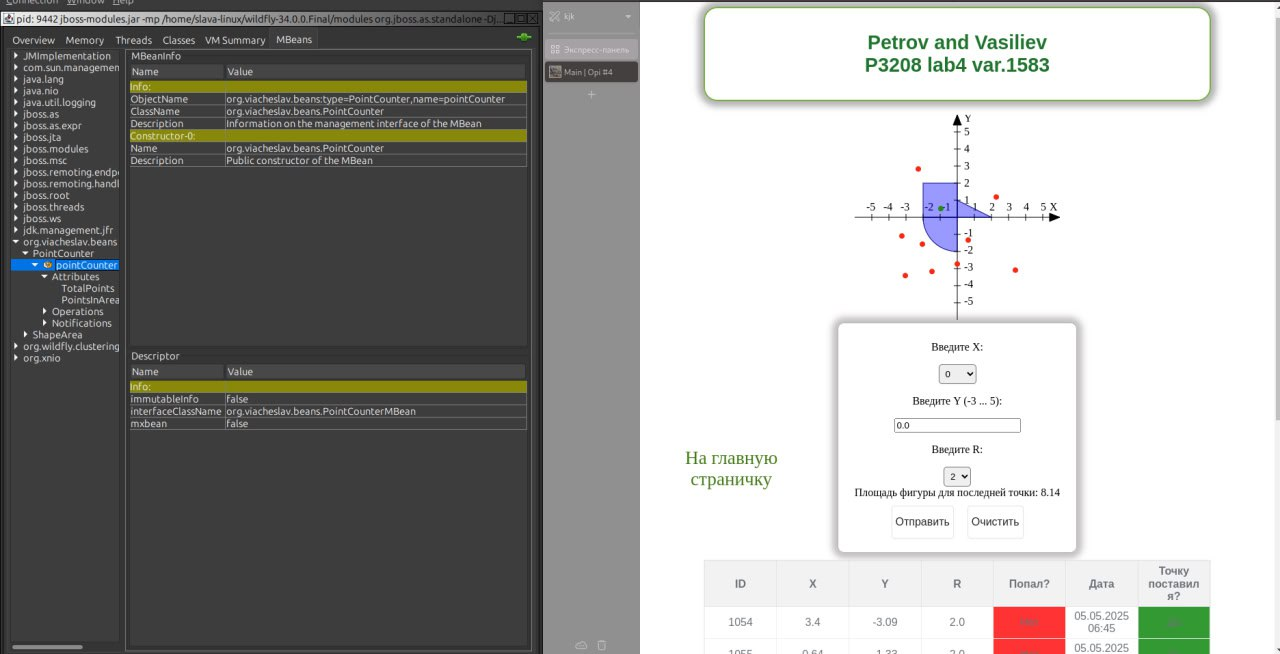
* Описание выявленной проблемы.
* Описание путей устранения выявленной проблемы.
* Подробное (со скриншотами) описание алгоритма действий, который позволил выявить и локализовать проблему.

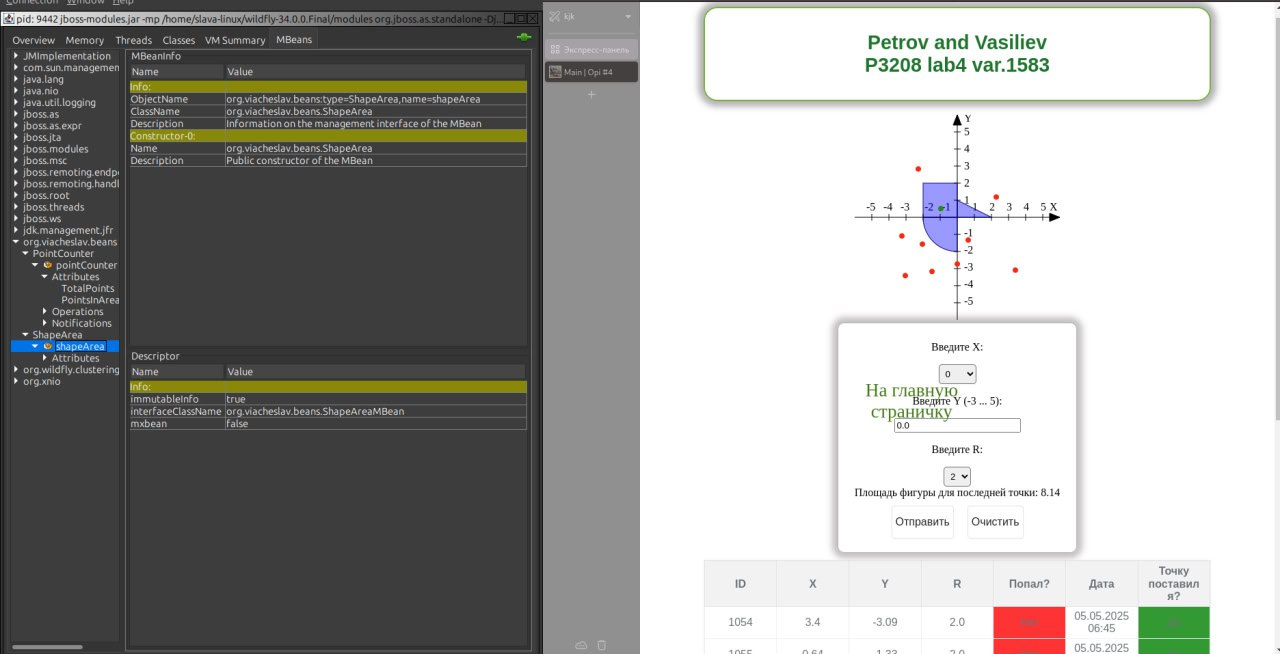
Код программы

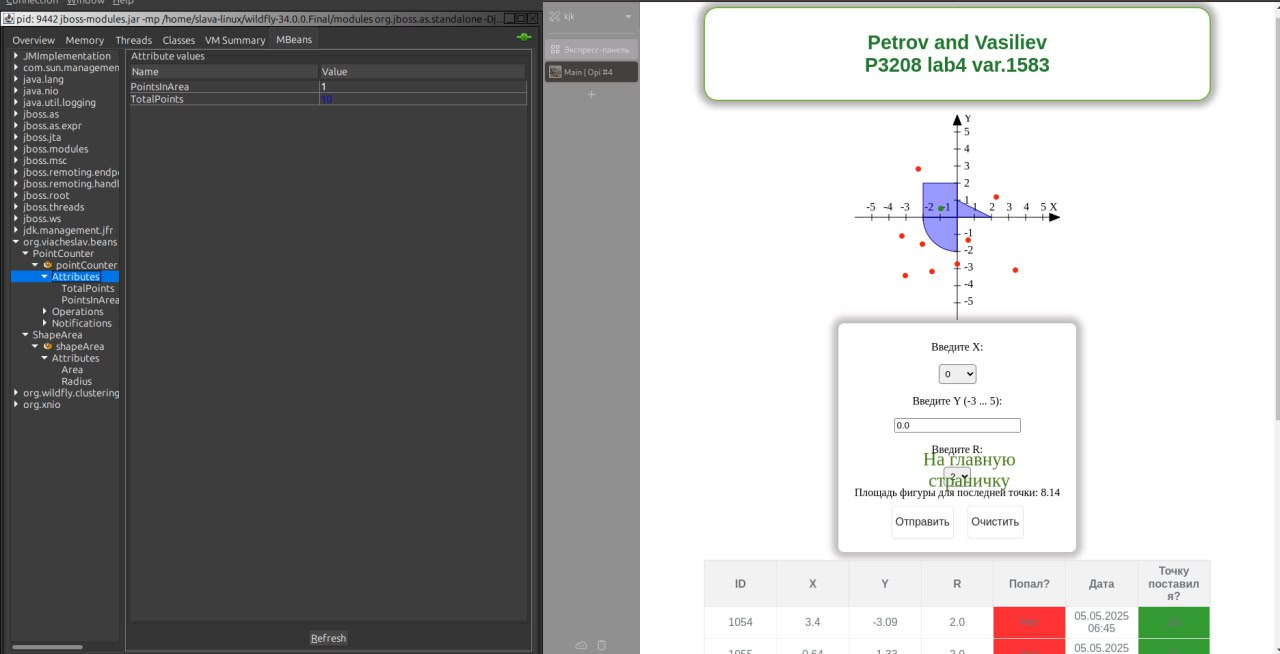
<https://github.com/kihort-si/opi-lab4>

Показания JConsole

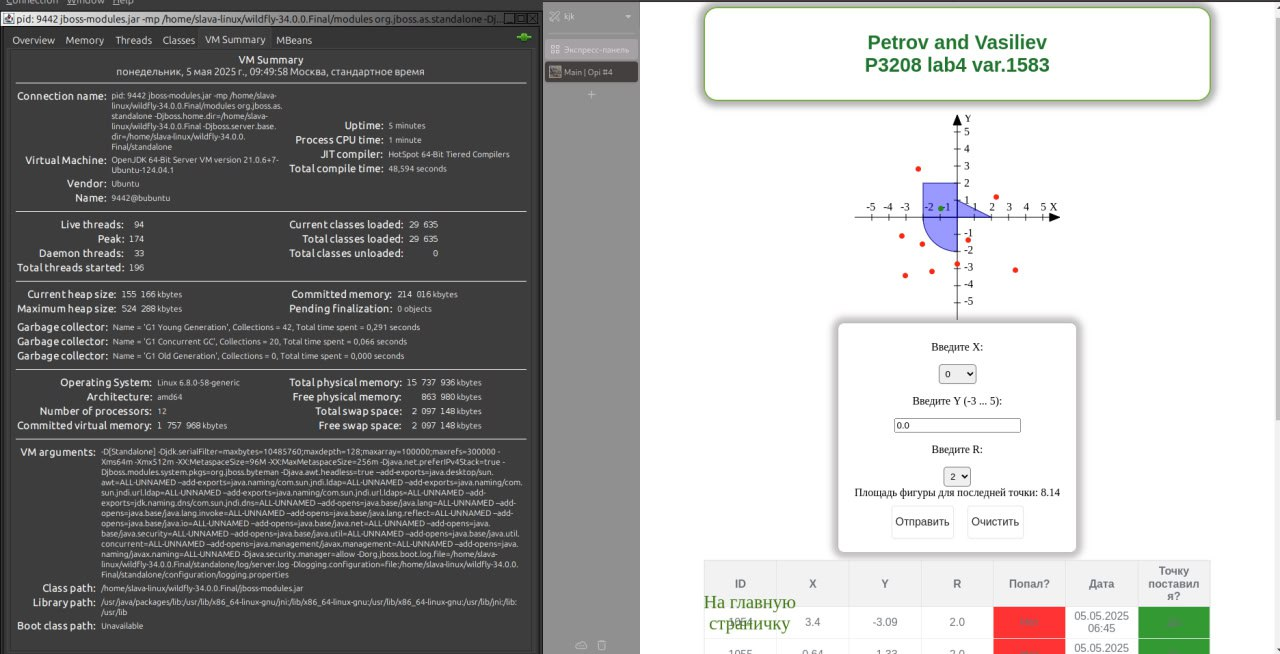
Показания MBean-классов, разработанных в ходе выполнения задания 1:

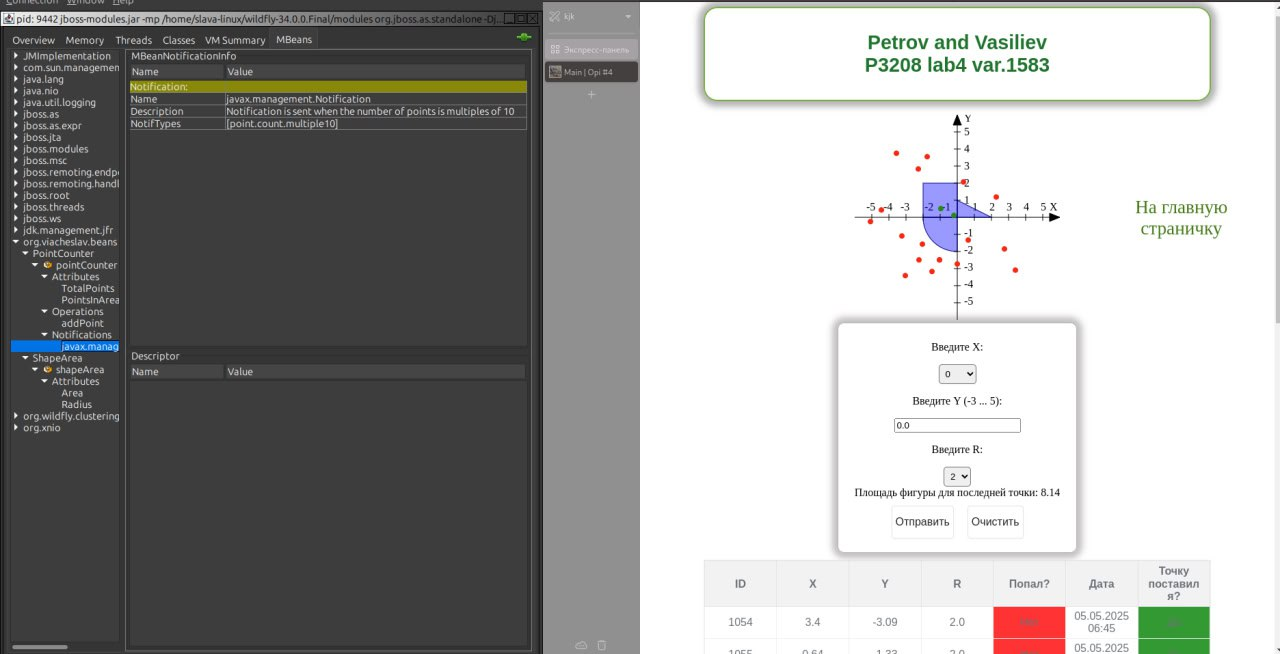












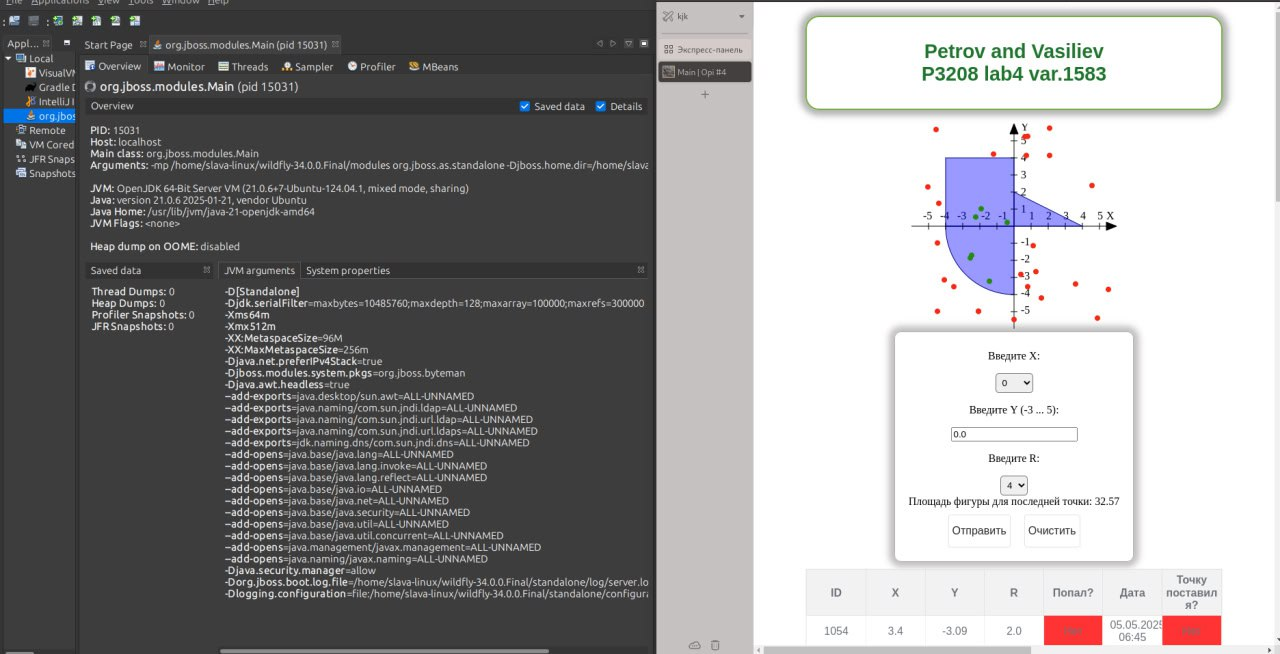
**Выводы по результатам мониторинга**:

* В процессе мониторинга с применением утилиты JConsole было установлено следующее:
* Для определения времени работы JVM можно воспользоваться показателем Uptime в разделе VM Stats.
* MBeans с именами PointCounter и ShapeArea были успешно созданы и зарегистрированы в системе.
* MBean PointCounter настроен на отправку уведомлений при десяти попаданиях. Эти уведомления корректно отображаются в JConsole.

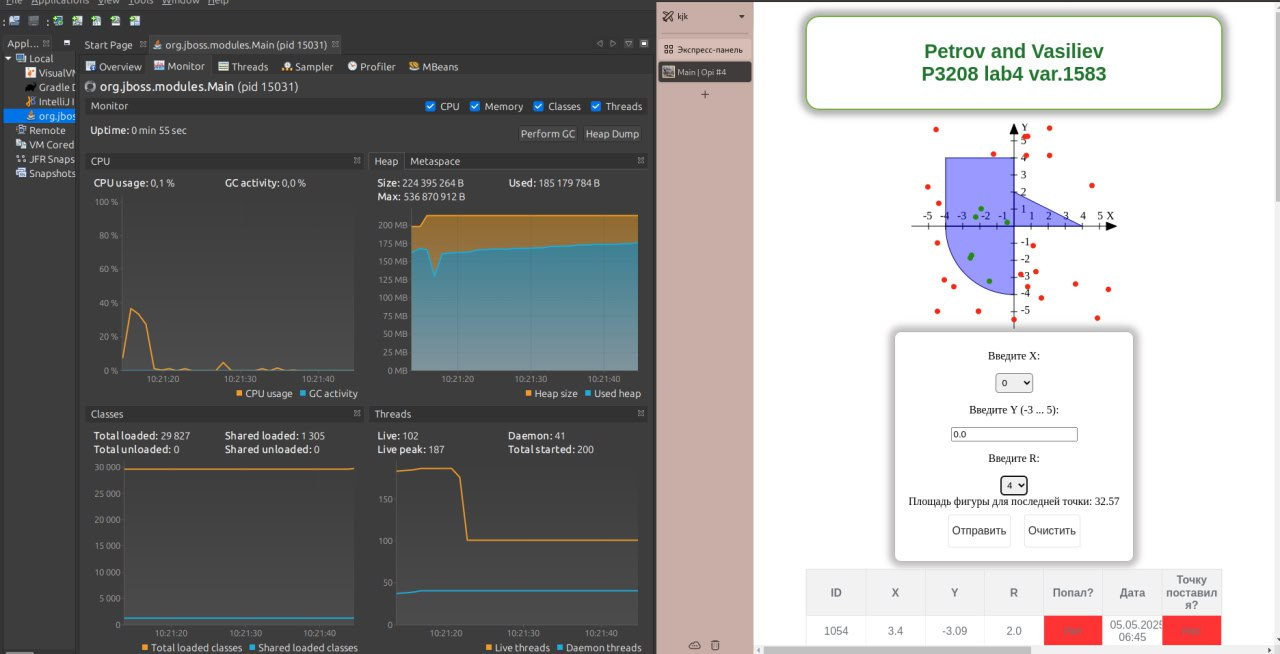
## Таким образом, механизм отправки и получения уведомлений от MBean был успешно реализован и протестирован, что обеспечивает возможность своевременного реагирования на события — ключевой аспект эффективного мониторинга.

Показания VisualVM

JVM Overview:

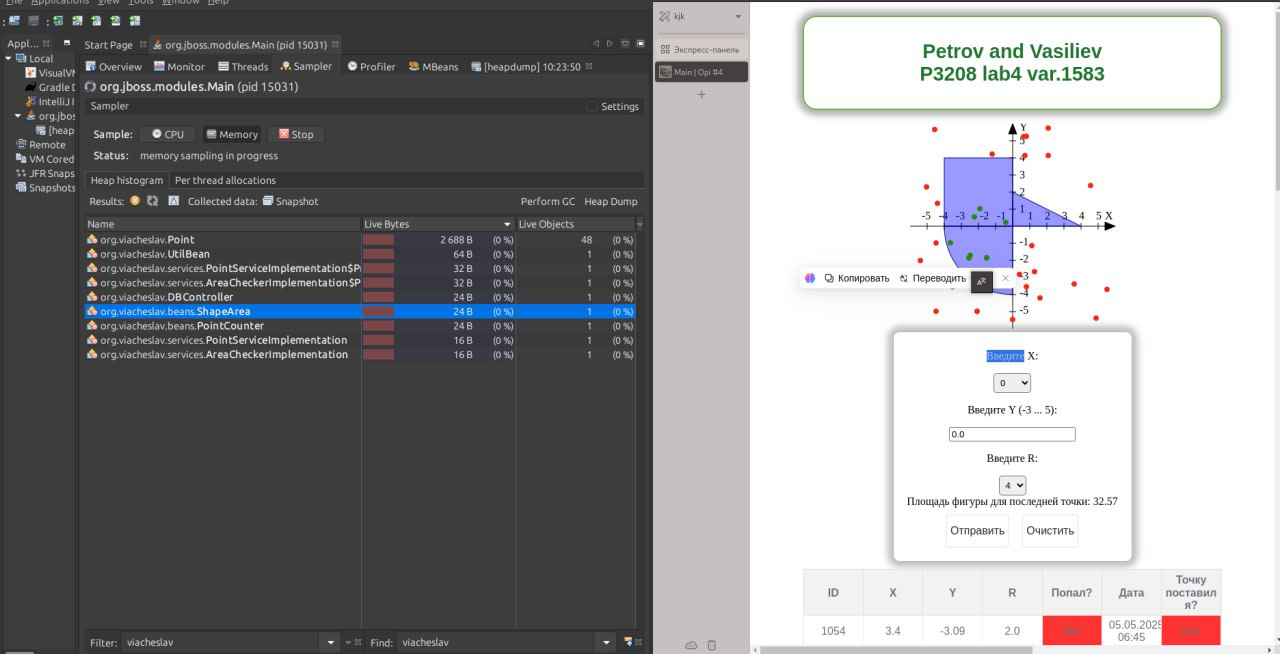


Мониторинг:

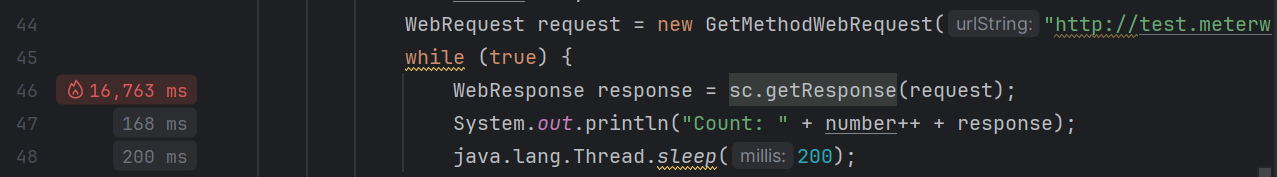


Определить имя класса, объекты которого занимают наибольший объём памяти JVM; определить пользовательский класс, в экземплярах которого находятся эти объекты.





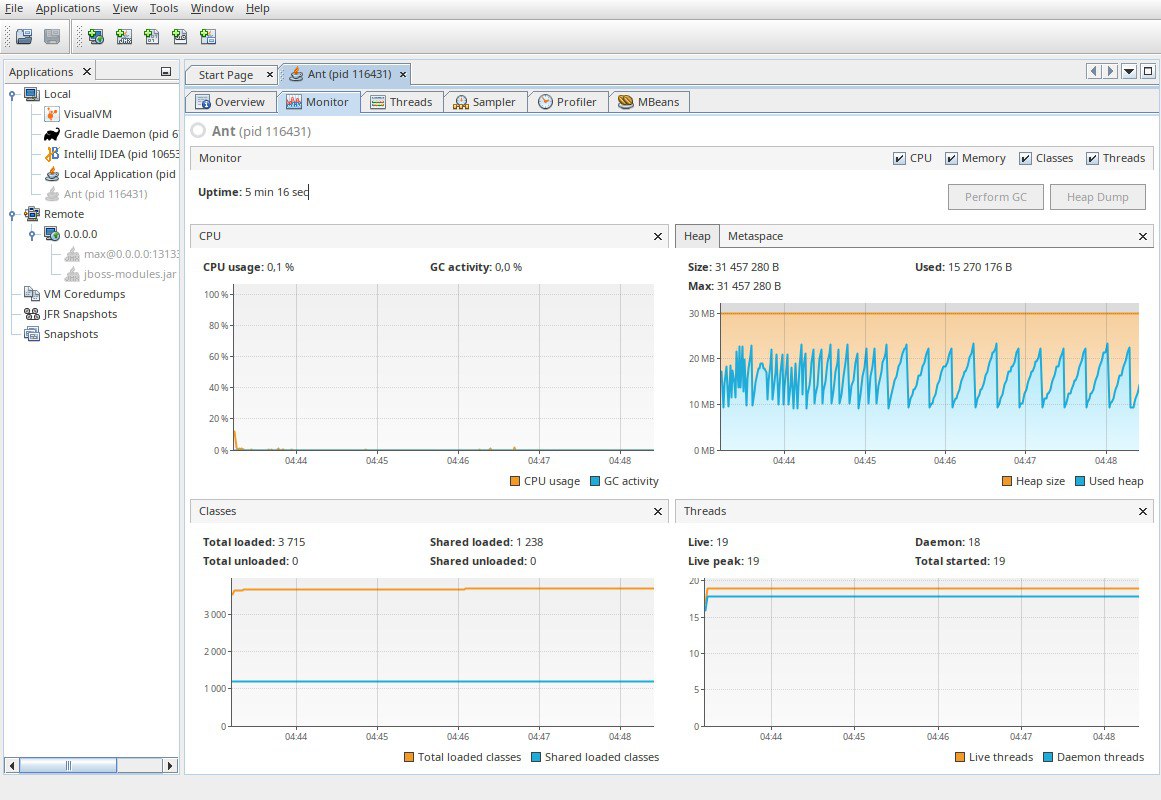
Исследования проблем с производительностью в программе HttpUnit

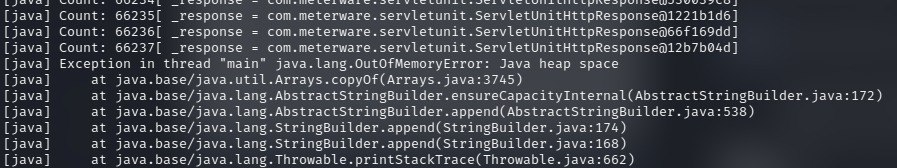


В ходе анализа производительности приложения была выявлена проблема, связанная с частыми вызовами метода java.lang.Thread.sleep(200). Этот метод приостанавливает выполнение потока на 200 миллисекунд, что влечёт за собой необоснованные задержки и снижает общую эффективность работы системы.

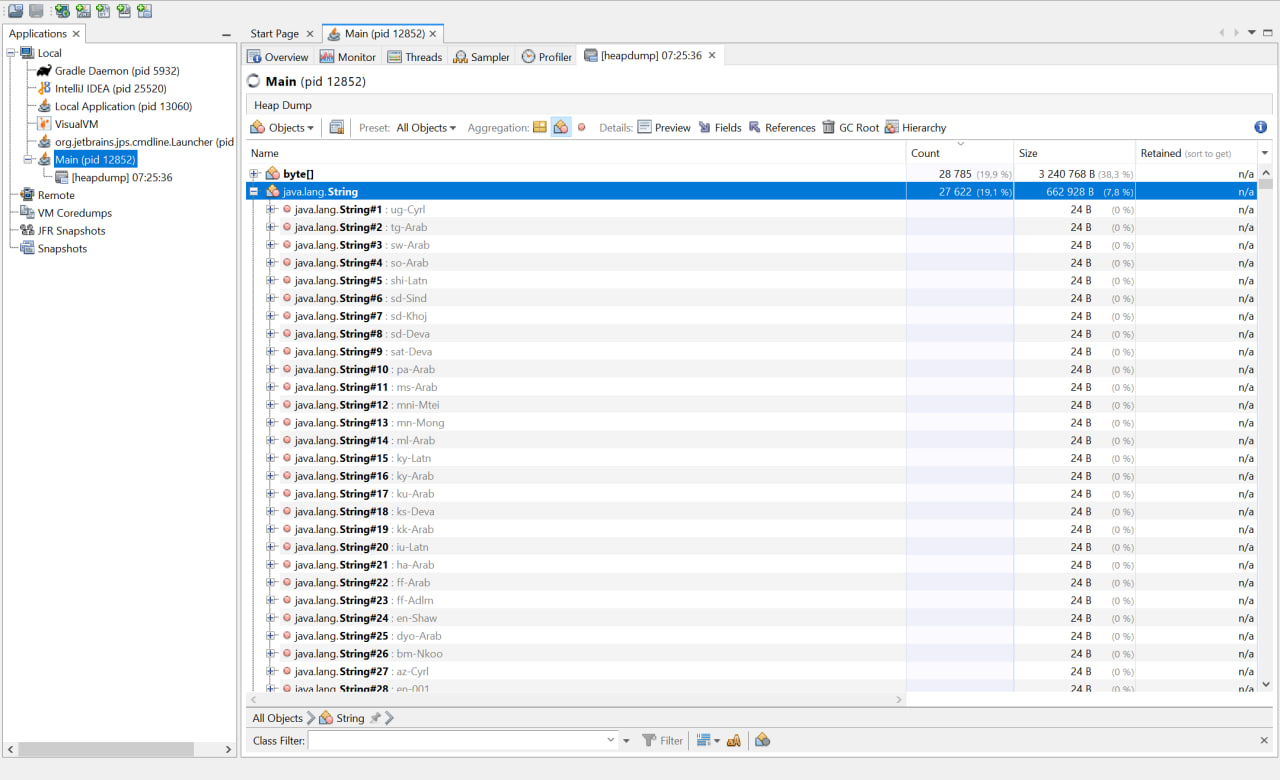
Поскольку данный вызов не выполняет полезных действий и не влияет на логику работы приложения, его можно безопасно удалить. Это позволит устранить лишние паузы и повысить производительность.

Зададим максимальный размер кучи равным 30 МБ с помощью параметра -Xmx30m, указав его при запуске программы через командную строку.





С помощью Heap Dump найдем объекты, занимающие большую часть памяти.





Исследовав кучу, находим повторяющиеся строки:

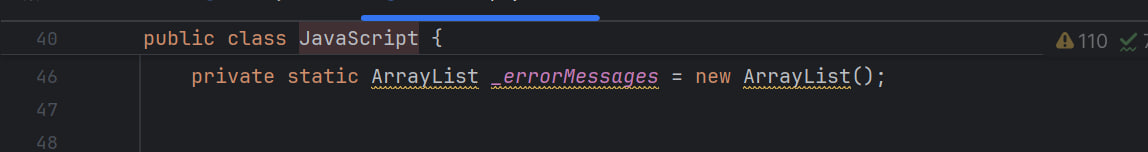
Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

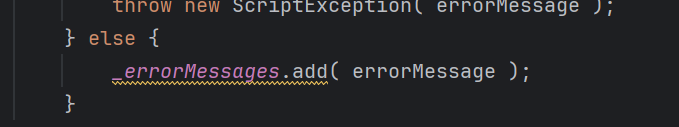
Изображение выглядит как текст, Шрифт, линия, снимок экрана

Автоматически созданное описание

Объекты \_errorMessages хранятся в ArrayList



Найдем строчку с добавлением объектов в этот список:



Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описаниеВ результате получается накопление \_errorMessages в списке, за счет чего и получается переполнение памяти. В программе есть функция для очистки \_errorMessage, однако мы можем увидеть, что на самом деле она не используется

Изображение выглядит как текст, Шрифт, рукописный текст, снимок экрана

Автоматически созданное описание

Изображение выглядит как текст, Шрифт, снимок экрана

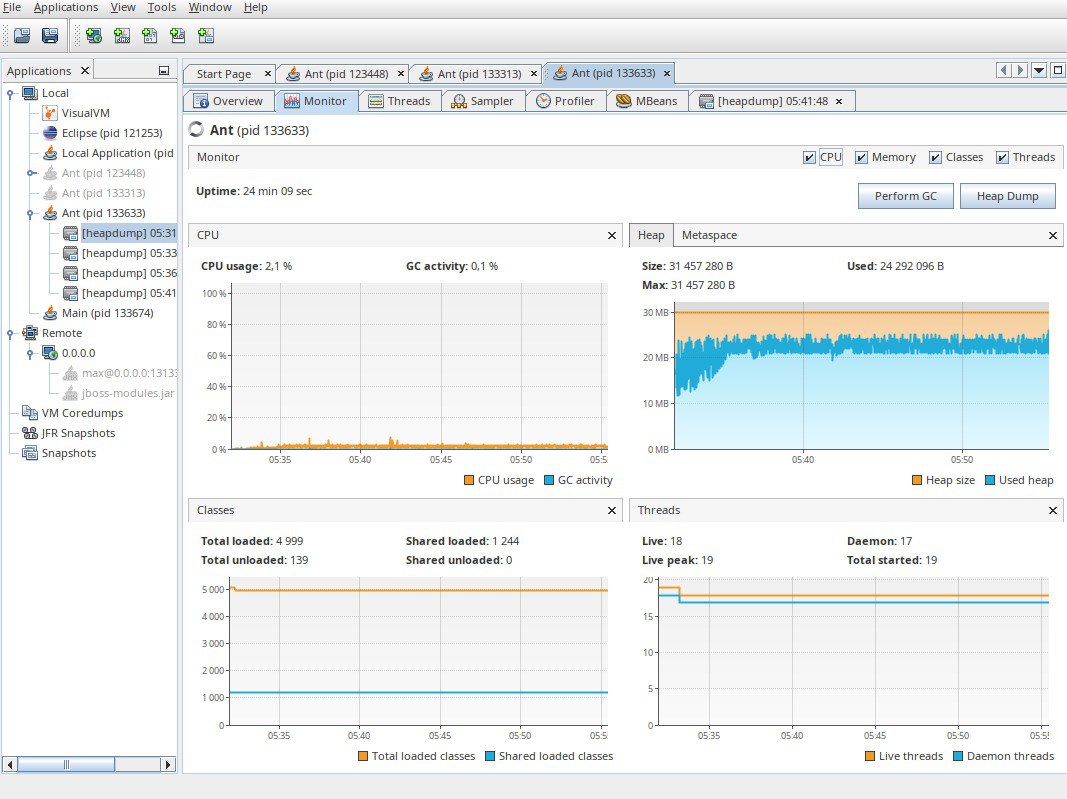
Автоматически созданное описание

Решением будет очистка списка с \_errorMessage после выполнения очередного запроса.

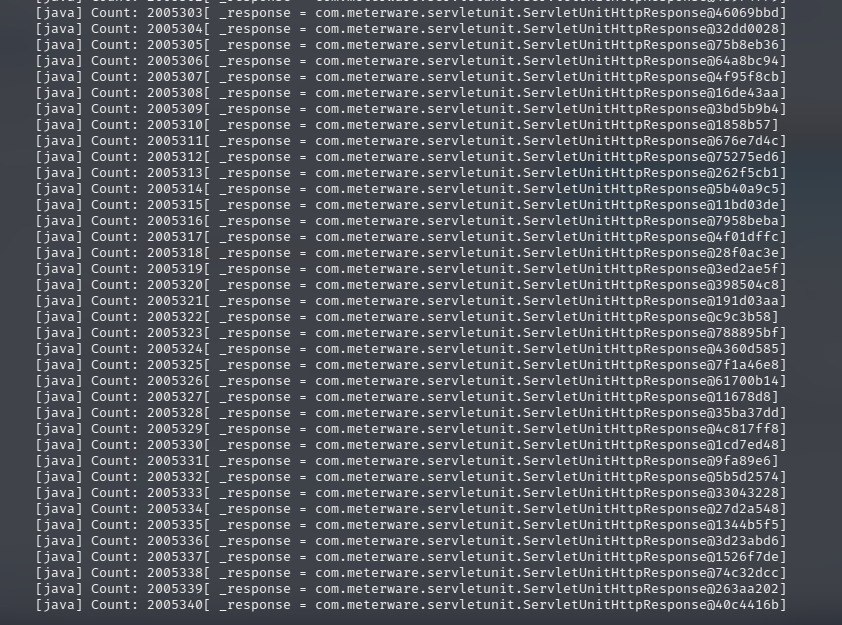
Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Запустим программу. Теперь изменения памяти во времени менее пилообразные, GC работает более оптимально.



Программа работает стабильно и не падает с OutOfMemoryException.



# Вывод

Во время выполнения лабораторной работы я познакомился с практикой написания MBeans в веб-приложениях, были изучены утилиты для мониторинга и профилирования работы программы JConsole и VisualVM, а также был получен опыт по полученным данным определять утечки памяти и устранять их.