**Министерство науки и высшего образования Российской Федерации**

федеральное государственное автономное образовательное учреждение

высшего образования

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

**Отчет**

по лабораторной работе № 4 по дисциплине «**Методы и средства программной инженерии**»

Вариант 1791

Автор: Иванов Андрей Вячеславович

Факультет: ПИиКТ

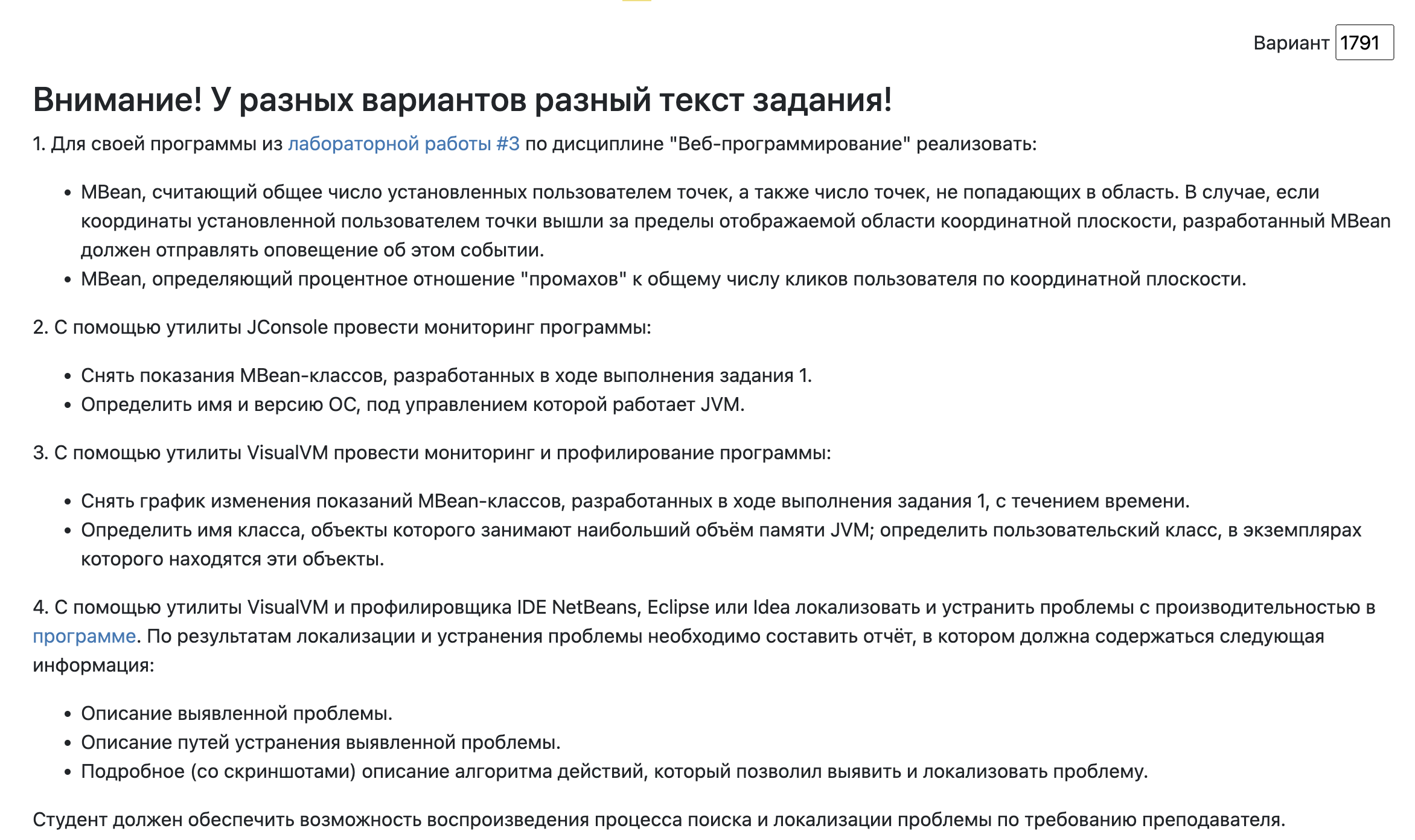
Группа: P32101

Преподаватель: Цопа Евгений Алексеевич



Санкт-Петербург, 2023

**Задание:**

****

**Выполнение:**

1. **Исходный код разработанных MBean-классов и сопутствующих классов.**

* MarkCounter

A screen shot of a computer program

Description automatically generated

A computer screen shot of text

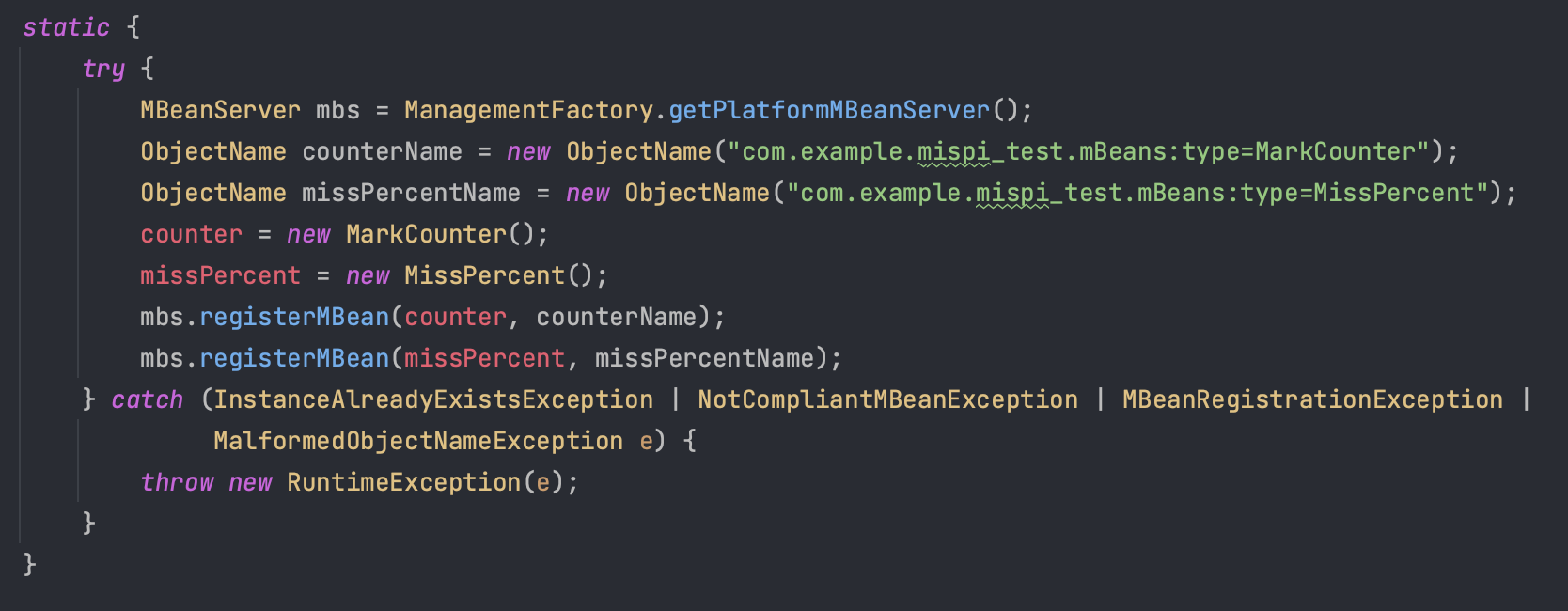
Description automatically generated

* MissPercent

A screenshot of a computer program

Description automatically generated

* MarkManager



1. **Скриншоты программы JConcole со снятыми показаниями, выводы по результатам мониторинга.**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

Имя и версия ОС, под управлением которой работает JVM:**A black text on a white background

Description automatically generated**

**Выводы:**

Был произведен мониторинг работы программы с помощью утилиты Jconsole. Утилита отслеживает изменения показаний MBean’ов, также позволяет вызывать операции и отслеживать уведомления. Также с помощью утилиты JConsole было определено имя и версия ОС, под управлением которой работает JVM.

1. **Скриншоты программы VisualVM со снятыми показаниями, выводы по результатам профилирования.**

**A screenshot of a computer

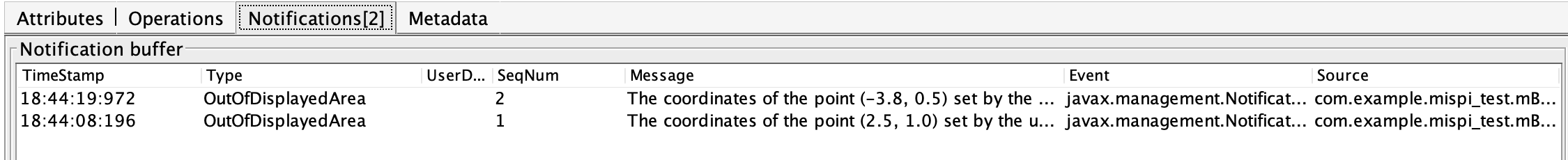
Description automatically generated**

A screenshot of a computer

Description automatically generated

A screenshot of a graph

Description automatically generated



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

A screenshot of a computer

Description automatically generated

Как можно заметить на скриншоте выше, ArrayList – класс объекты которого занимают наибольший объём памяти JVM. MarkDTO – пользовательский класс, в экземплярах которого находятся эти объекты. Более того можно заметить, что экземпляры класса Mark занимают сравнительно такое же количество памяти JVM. Данная ситуация связанна с тем, что класс MarkDTO выступает в роли обёртки для класса Mark. Это сделано для того, чтобы пользователь мог видеть время своего часового пояса, так как в базе данных время хранится в UTC. Объекты, хранящиеся в ArrayList занимают наибольший объём памяти JVM, так как для отображения точек на графике и в таблице я достаю точки из базы и заполняю ими список.

**Выводы:**

Были произведены мониторинг и профилирование работы программы с помощью утилиты VisualVM. Утилита отслеживает изменения показаний MBean’ов, позволяет вывести графики изменения показаний MBean’ов с течением времени и отслеживать уведомления. Также с помощью утилиты VisualVM было определено имя класса, объекты которого занимают наибольший объём памяти JVM и пользовательского класса, в экземплярах которого находятся эти объекты.

1. **Скриншоты программы VisualVM с комментариями по ходу поиска утечки памяти.**

* **Анализ программы и описание выявленной проблемы**

Сначала заглянем в класс Main – главный класс нашего приложения. Там можно увидеть основную часть программы – бесконечный цикл с запросами.

A computer code with text

Description automatically generated

Чтобы определить проблему попробуем уменьшить задержку между запросами путём установки Tread.sleep(0) в 0 мс.

По графику использования памяти видно, что размер кучи постоянно увеличивается:

A graph with blue and orange lines

Description automatically generated

Установим максимальный размер кучи равным 25 Мб. Для этого в среде разработки IntelliJ IDEA в конфигурации запуска для VM добавим параметр

-Xmx25m.

Запустим программу снова. Через пару минут после запуска программа выбрасывает исключение java.lang.OutOfMemoryError:

A screenshot of a computer program

Description automatically generated

По графику использования памяти видно, что размер кучи постоянно растёт:

A graph with blue and orange lines

Description automatically generated

Даже после отработки GC тенденция к росту остается. По графику Used Heap также можно заметить частый вызов GC, это негативно влияет на производительность.

A graph of a graph

Description automatically generated with medium confidence

Следовательно, можно сделать вывод о том, что в программе существует утечка памяти.

* **Локализация проблемы**

Проанализируем Heap Dump и найдём объекты, занимающие больше всего памяти при выполнении программы:

A screenshot of a computer

Description automatically generated

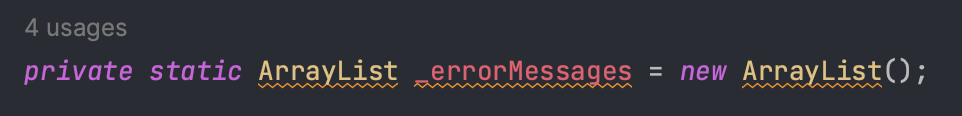
Заметим, что больше всего памяти занимают объекты типа char. Для того, чтобы понимать откуда у нас в программе столько объектов типа char, переключимся в раздел instances и увидим, что у нас есть объект ArrayList под названием \_errorMessages в котором хранится большое количество объектов типа String c информацией об ошибках.A screenshot of a computer

Description automatically generated

Объект \_errorMessages находится в классе com.meterware.httpunit.javascript.JavaScript.

Перейдём в данный класс и проанализируем код этого класса.

В данном классе находится список, в который записываются ошибки, а также метод, который добавляет ошибки в список:



A screen shot of a computer code

Description automatically generated

Более того в данном классе реализован метод для очистки списка ошибок, но. он нигде не используется.

A screen shot of a computer

Description automatically generated

То есть ошибки записываются, но нигде не удаляются, поэтому количество памяти, которое использует программа, растёт.

* **Устранение выявленной проблемы**

Для устранения выявленной проблемы можно проверять размер списка и при определённом количестве элементов очищать список.

Финальный метод для очистки списка находится в классе com.meterware.httpunit.HttpUnitOptions:

A screen shot of a computer error

Description automatically generated

Более того в классе com.meterware.httpunit.HttpUnitOptions существует метод, который возвращает накопленные сообщения об ошибках скрипта:

A screenshot of a computer program

Description automatically generated

Добавим очистку списка при определённой заполненности в главном цикле программы:

A screen shot of a computer code

Description automatically generated

A graph with blue lines

Description automatically generated

A graph with blue and orange lines

Description automatically generated

После запуска исправленной программы наблюдаем, что теперь не расходуется больше 25 Мб кучи, тенденции к росту размера кучи не наблюдается и сборщик мусора работает в нормальном режиме. Более того программа начала работать. быстрее, так как уменьшилась активность GC.

**Вывод:**

В ходе выполнения данной лабораторной работы я ознакомился с профилированием Java приложений, изучил утилиты JConsole и VisualVM для мониторинга и профилирования Java-приложений. Поработал с технологией JMX, а именно создал свои MBean’ы для сбора статистики. Так же с помощью утилиты VisualVM я определил, локализовал и устранил проблемы в представленной Java программе. Всё это помогло мне разобраться в деталях работы JVM и научиться определять, как и какие компоненты приложения влияют на производительность.