

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: 200522

Преподаватель:

Харитонов Анастасия Евгеньевна

Выполнили:

Медведев Владислав Александрович

Михайлов Дмитрий Андреевич

Группа: P3206

Санкт-Петербург, 2025

1. Задание

Лабораторная работа #4

Вариант 200522

Внимание! У разных вариантов разный 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, не попадающих в область. В случае, если координаты установленной пользователем точки вышли за пределы отображаемой области координатной плоскости, разработанный MBean должен отправлять оповещение об этом событии.
- MBean, определяющий средний интервал между кликами пользователя по координатной плоскости.

2. С помощью утилиты JConsole провести мониторинг программы:

- Снять показания MBean-классов, разработанных в ходе выполнения задания 1.
- Определить имя и версию ОС, под управлением которой работает JVM.

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 объем 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 отчет, 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.

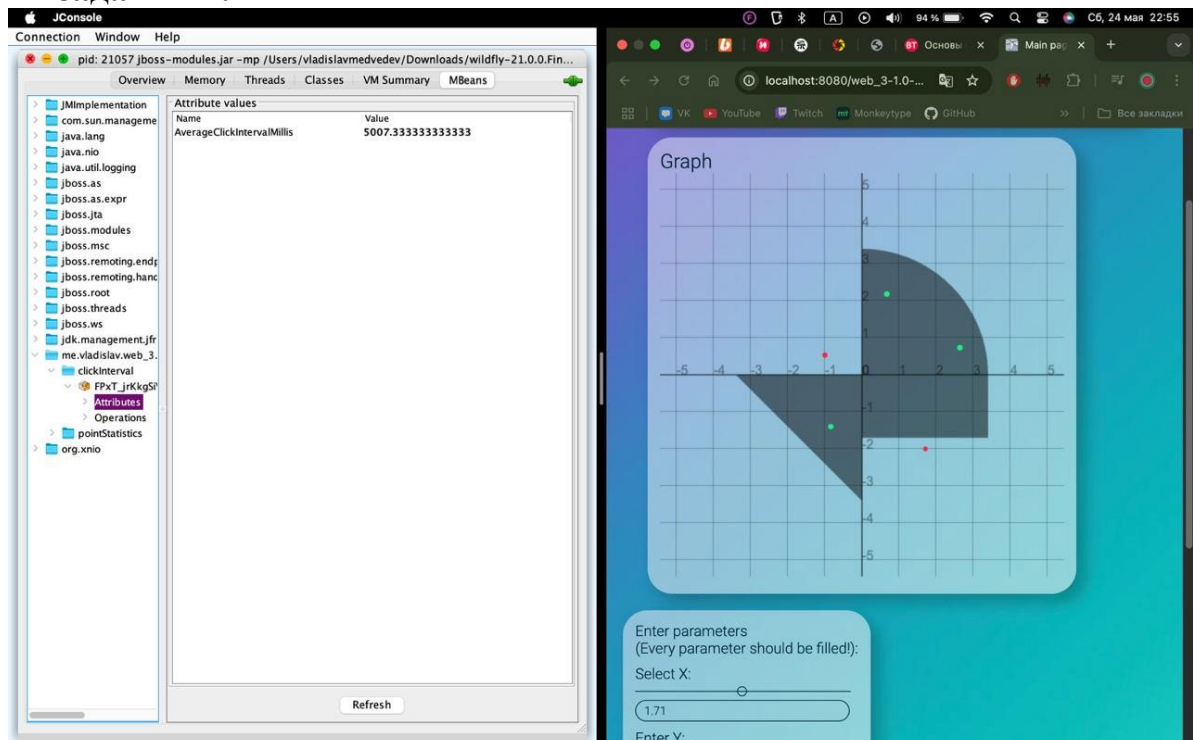
Студент должен обеспечить возможность воспроизведения процесса поиска и локализации проблемы по требованию преподавателя.

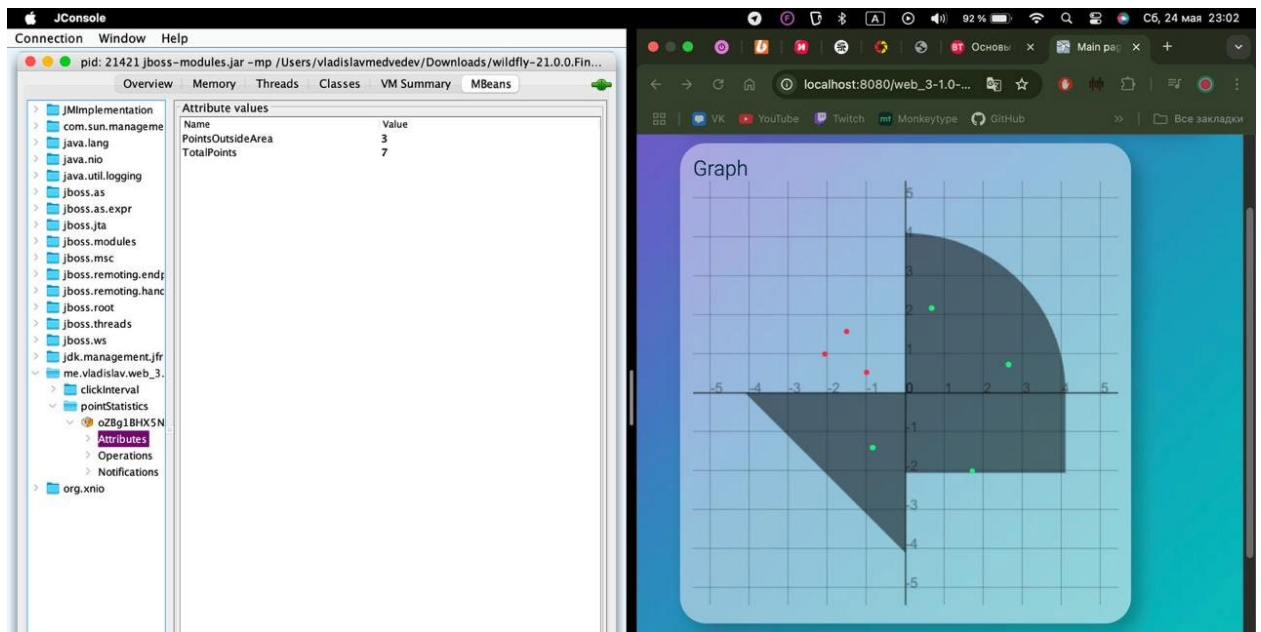
2. Выполнение

https://github.com/medvedev888/Web_3

3. Скриншоты программы JConsole

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





The bottom screenshot shows the JConsole application for PID 21230. The 'MBeans' tab is selected, and the tree on the left shows the hierarchy: `org.jboss.as` > `org.jboss.as.web` > `me.vladislav.web_3` > `clickInterval` > `pointStatistics` > `EZ3_kLWhKZl` > `Attributes` > `Notifications`. The 'Notification buffer' table on the right shows the following data:

TimeStamp	Type	UserData	SeqNum	Message	Event	Source
22:57:30:777	me.vladislav....		2	Точка вне об...	javax.manag...	me.vladislav.web...
22:57:27:231	me.vladislav....		1	Точка вне об...	javax.manag...	me.vladislav.web...

- **Имя и версию ОС, под управлением которой работает JVM:**

pid: 21421 jboss-modules.jar -mp /Users/vladislavmedvedev/Downloads/wildfly-21.0.0.Final/modules org.jboss.as.standalone -Djboss.home.dir=/Users/vladislavmedvedev/Downloads/wildfly-21.0.0.Final ...	
Overview Memory Threads Classes VM Summary MBeans	
VM Summary суббота, 24 мая 2025 г., 23:02:50 Москва, стандартное время	
Connection name: pid: 21421 jboss-modules.jar -mp /Users/vladislavmedvedev/Downloads/wildfly-21.0.0.Final/modules org.jboss.as.standalone -Djboss.home.dir=/Users/vladislavmedvedev/Downloads/wildfly-21.0.0.Final -Djboss.server.base.dir=/Users/vladislavmedvedev/Downloads/wildfly-21.0.0.Final/standalone Virtual Machine: OpenJDK 64-Bit Server VM version 11.0.25+9-LTS Vendor: Amazon.com Inc. Name: 21421@MacBook-Air-Vladislav.local	Uptime: 2 minutes Process CPU time: 59,964 seconds JIT compiler: HotSpot 64-Bit Tiered Compilers Total compile time: 34,236 seconds
Live threads: 87 Peak: 169 Daemon threads: 35 Total threads started: 210	Current classes loaded: 26 908 Total classes loaded: 26 945 Total classes unloaded: 37
Current heap size: 134 024 kbytes Maximum heap size: 524 288 kbytes Garbage collector: Name = 'G1 Young Generation', Collections = 89, Total time spent = 0,802 seconds Garbage collector: Name = 'G1 Old Generation', Collections = 0, Total time spent = 0,000 seconds	Committed memory: 200 784 kbytes Pending finalization: 0 objects
Operating System: Mac OS X 15.0 Architecture: aarch64 Number of processors: 8 Committed virtual memory: 413 393 184 kbytes	Total physical memory: 8 388 608 kbytes Free physical memory: 90 064 kbytes Total swap space: 2 097 152 kbytes Free swap space: 1 480 960 kbytes
VM arguments: -D[Standalone] -Xms64m -Xmx512m -XX:MetaspaceSize=96M -XX:MaxMetaspaceSize=256m -Djava.net.preferIPv4Stack=true -Djboss.modules.system.pkgs=org.jboss.byteman -Djava.awt.headless=true --add-exports=java.base/sun.nio.ch=ALL-UNNAMED --add-exports=jdk.unsupported/sun.misc=ALL-UNNAMED --add-exports=jdk.unsupported/sun.reflect=ALL-UNNAMED -Dorg.jboss.boot.log.file=/Users/vladislavmedvedev/Downloads/wildfly-21.0.0.Final/standalone/log/server.log -Dlogging.configuration=file:/Users/vladislavmedvedev/Downloads/wildfly-21.0.0.Final/standalone/configuration/logging.properties Class path: /Users/vladislavmedvedev/Downloads/wildfly-21.0.0.Final/jboss-modules.jar Library path: /Users/vladislavmedvedev/Library/Java/Extensions/Library/Java/Extensions:/Network/Library/Java/Extensions:/System/Library/Java/Extensions:/usr/lib/java: Boot class path: Unavailable	

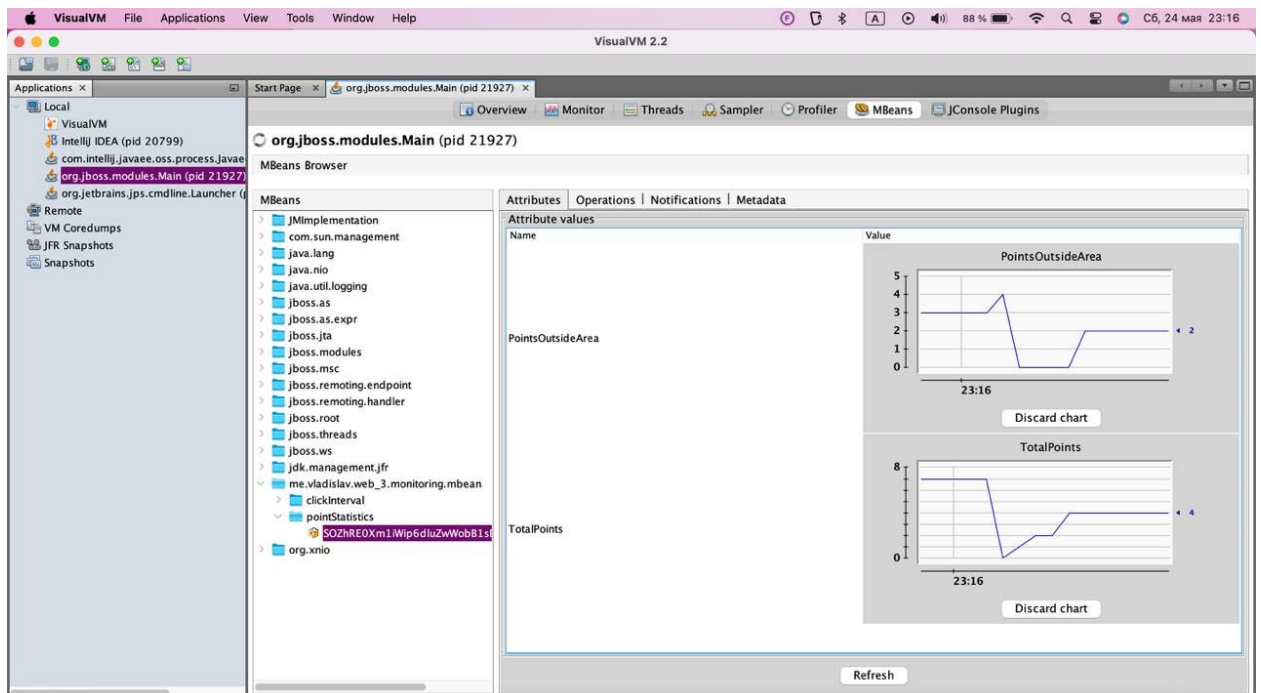
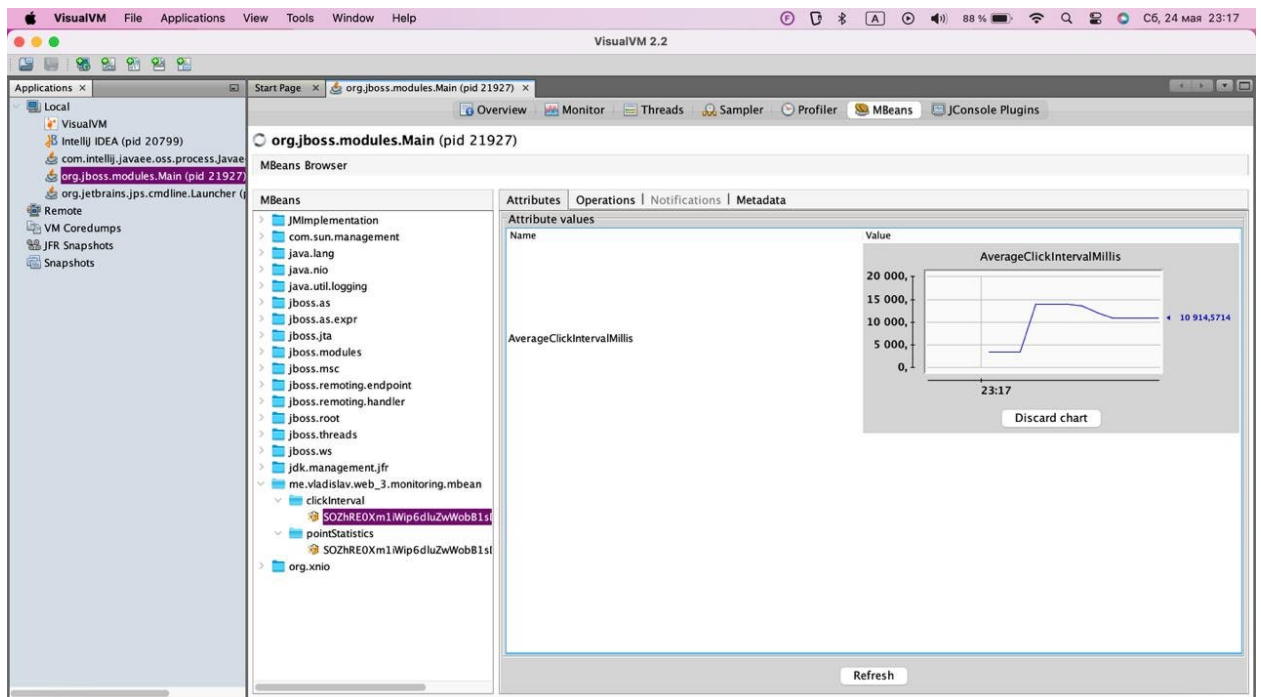
• Выводы по результатам мониторинга:

В ходе мониторинга с использованием утилиты JConsole было определено, что:

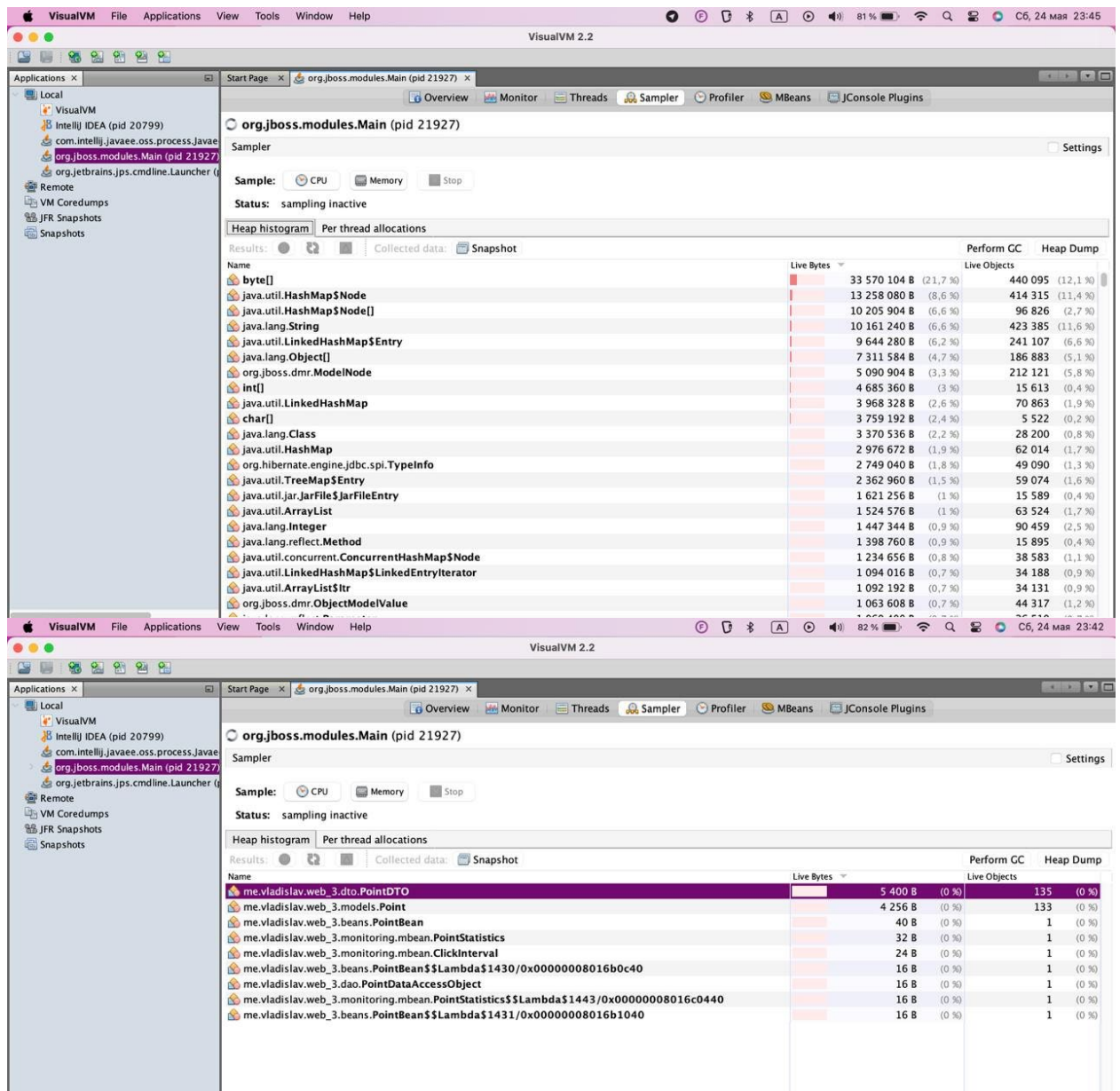
- Вкладка VM Summary в JConsole показывает сводную информацию о виртуальной машине Java (JVM)
- MBean ClickInterval и PointStatistics были успешно разработаны и зарегистрированы. В случае промаха MBean Point Statistic отправляет уведомление. Таким образом, было зарегистрировано и протестировано получение уведомлений от MBean, что позволяет оперативно реагировать на события, что является важной частью мониторинга.

4. Скриншоты программы VisualVM

- График изменения показаний MBean-классов, разработанных в ходе выполнения задания 1, с течением времени



- Имя класса, объекты которого занимают наибольший объём памяти JVM; пользовательский класс, в экземплярах которого находятся эти объекты



- **Выводы по результатам мониторинга и профилирования:**

Изменения показаний MBean-классов с течением времени:

На графиках ClickInterval и PointStatistics видны изменения количества точек, а также количество промахов. График PointStatistics показывает динамику добавления попавших и не попавших точек. График ClickInterval показывает динамику изменения среднего времени между нажатиями на график для добавления точек.

Определение имени класса, объекты которого занимают наибольший объём памяти JVM; определение пользовательского класса, в экземплярах которого находятся эти объекты:

На основе профилирования памяти видно, что больше всего памяти занимают значения `byte[]`, на втором месте `HashMap$Node` – всего эти объекты занимают примерно 30%. Больше всего памяти из пользовательских классов занимают объекты `PointDTO` – 55% от пользовательских классов. Однако в масштабе всего приложения эти объекты занимают почти 0%.

5. Исследование программы

- Избыточный вызов метода

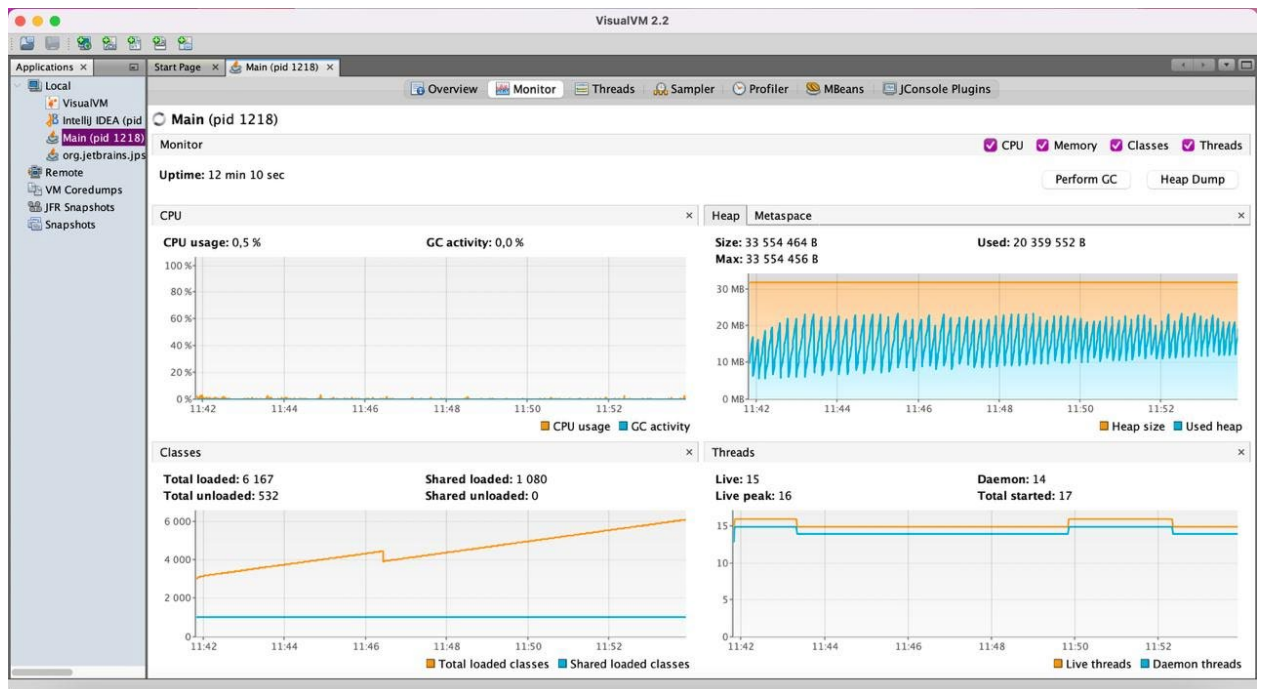


```
37 public static void main(String[] args) {
38     try {
39         HttpUnitOptions.setExceptionsThrownOnScriptError(false);
40         ServletRunner sr = new ServletRunner();
41         sr.registerServlet( resourceName: "myServlet", HelloWorld.class.getName());
42         ServletUnitClient sc = sr.newClient();
43         int number = 1;
44         WebRequest request = new GetMethodWebRequest( urlString: "http://test.meterware.com/myServlet");
45         while (true) {
46             WebResponse response = sc.getResponse(request);
47             System.out.println("Count: " + number++ + response);
48             java.lang.Thread.sleep( millis: 200);
49         }
50     }
51 }
```

Посмотрев на скрины, мы можем увидеть, что у нас поток `main` 98% времени находится в состоянии `sleeping`. Эта проблема связана с избыточным вызовом метода `java.lang.Thread.sleep(200)`. Данный метод вызывает приостановку выполнения потока на 200 миллисекунд, что приводит к задержкам и снижению общей производительности системы.

- **Переработка GC и утечка памяти**

Установив максимальный размер кучи в 30Мб с помощью -Xmx30m и запустим программу и посмотрим, как она себя ведет, в VisualVM.

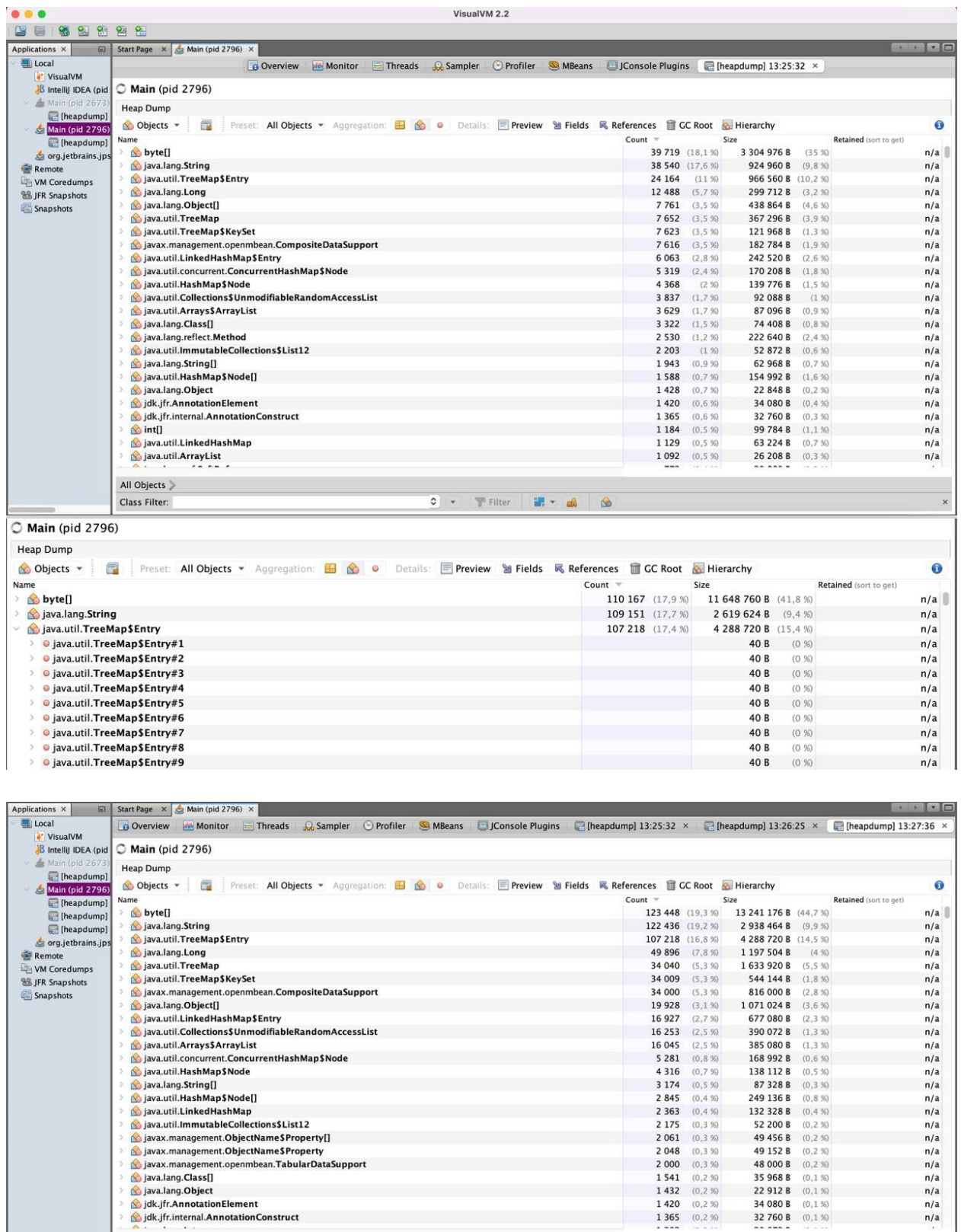


Мы можем заметить 2 проблемы: первая – переработка GC; вторая – утечка памяти (примерно каждые 7-10 секунд), так как после сборки мусора памяти в heap становится меньше, чем при предыдущих сборках мусора. Было 6900В, стало доходить до 12000В.

Убрав задержку и запустив снова программу, мы можем видеть, что программа прекратила свое выполнение примерно через 3 минуты из-за ошибки `java.lang.OutOfMemoryError`. Это также подтверждает наличие проблемы утечки памяти.

```
Count: 192907[ _response = com.meterware.servletunit.ServletUnitHttpResponse@6e9e3708]
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
    at java.base/java.io.ByteArrayOutputStream.<init>(ByteArrayOutputStream.java:81)
    at java.base/java.io.ByteArrayOutputStream.<init>(ByteArrayOutputStream.java:66)
    at org.mozilla.classfile.ConstantPool.addUtf8(ClassFileWriter.java:1412)
    at org.mozilla.classfile.ConstantPool.addMethodRef(ClassFileWriter.java:1488)
    at org.mozilla.classfile.ClassFileWriter.add(ClassFileWriter.java:814)
    at org.mozilla.javascript.optimizer.Codegen.addScriptRuntimeInvoke(Codegen.java:222)
    at org.mozilla.javascript.optimizer.Codegen.generatePrologue(Codegen.java:1472)
    at org.mozilla.javascript.optimizer.Codegen.generateCode(Codegen.java:501)
    at org.mozilla.javascript.optimizer.Codegen.compile(Codegen.java:97)
    at org.mozilla.javascript.Context.compile(Context.java:2076)
    at org.mozilla.javascript.Context.compile(Context.java:2006)
```

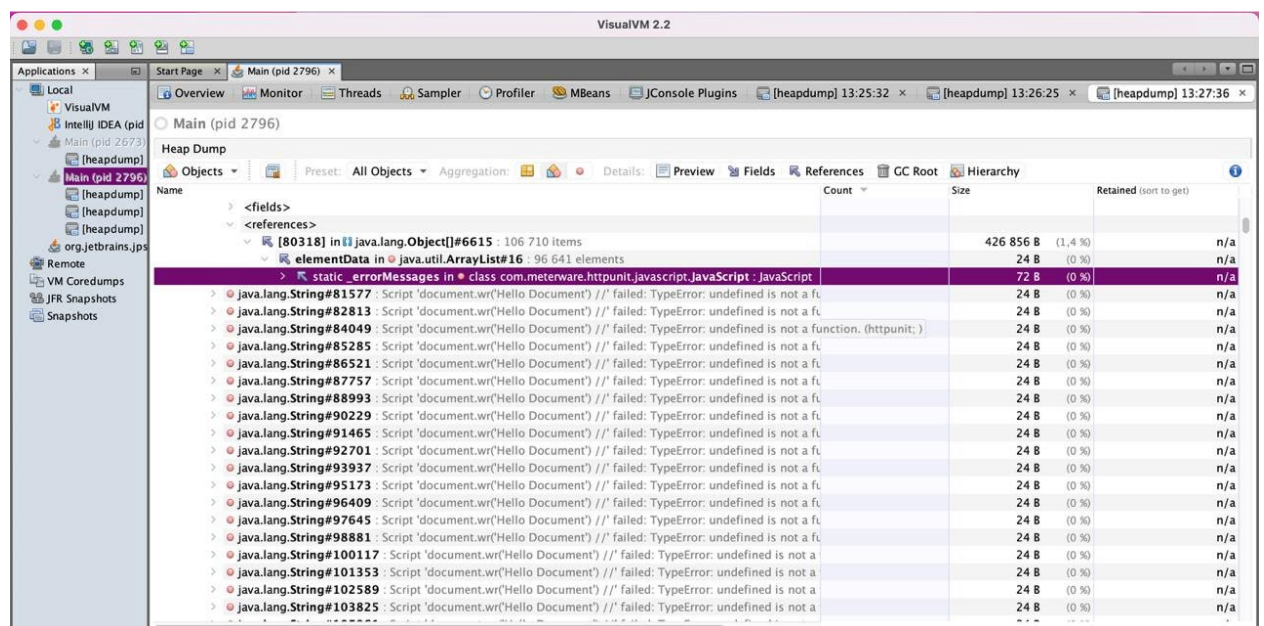
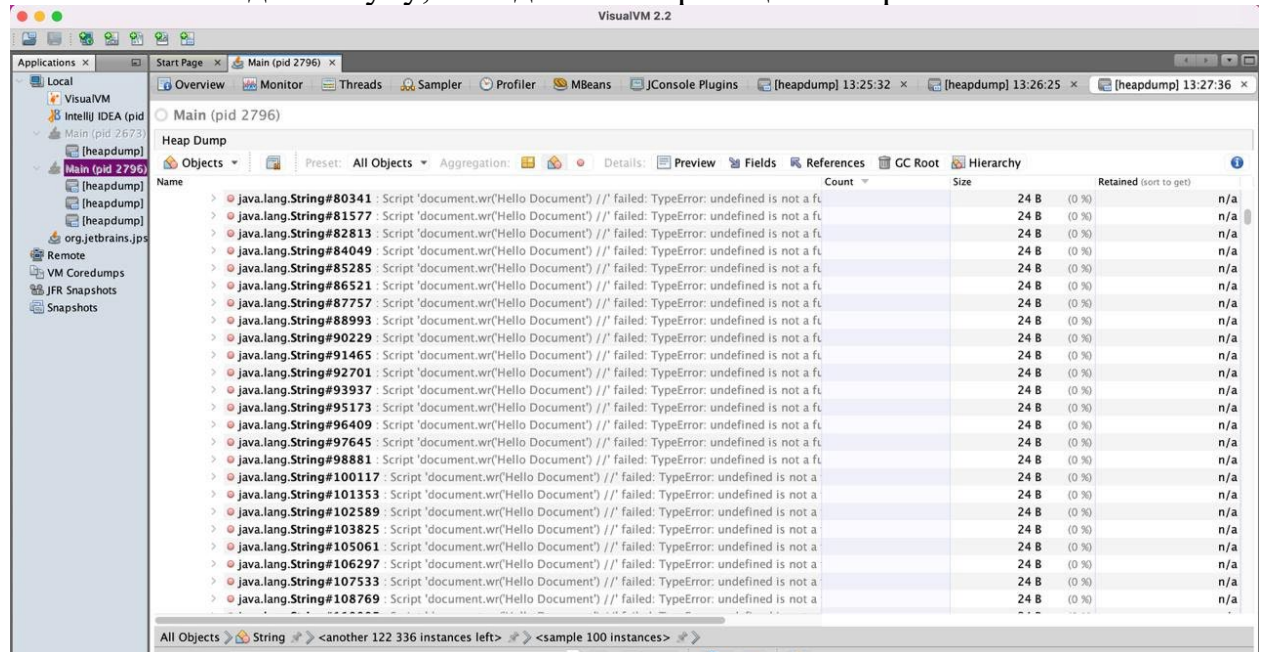
С помощью Heap Dump найдем объекты, занимающие большую часть памяти.



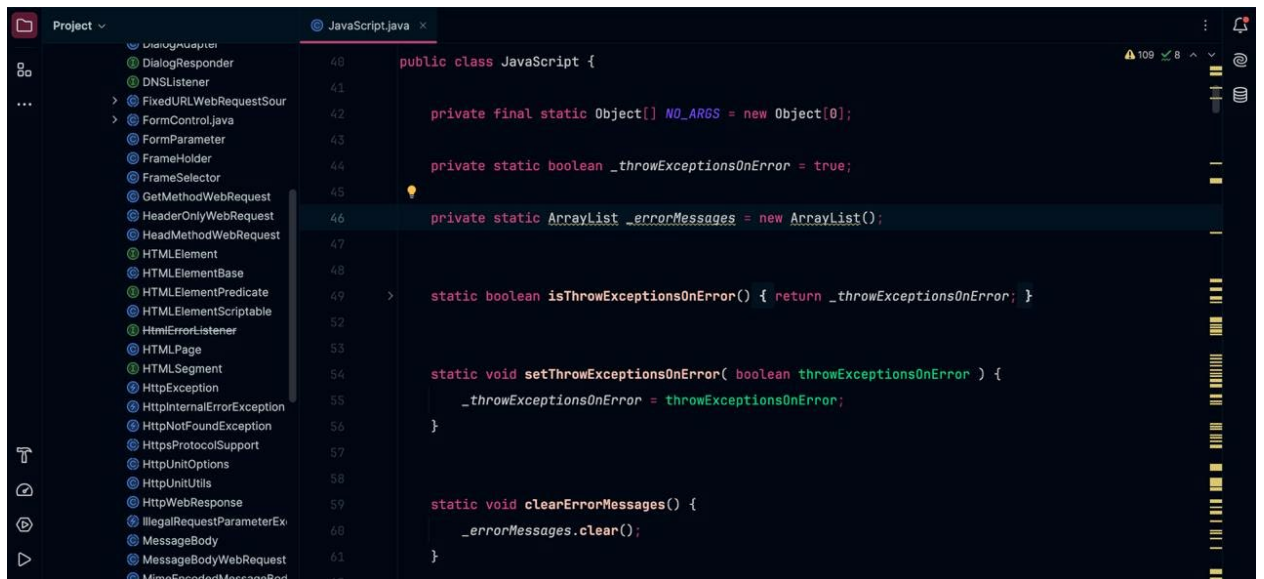
Из графиков использования памяти видно, что при работе приложения на каждый запрос создаются массивы `byte[]`, затем используются по всему приложению и остаются использованными, но не очищенными в памяти. GC отрабатывает корректно, но он тратит слишком много ресурсов на

очистке памяти, поэтому чтобы убрать излишнюю нагрузку надо устранить утечку.

Немного исследовав кучу, находим повторяющиеся строки:



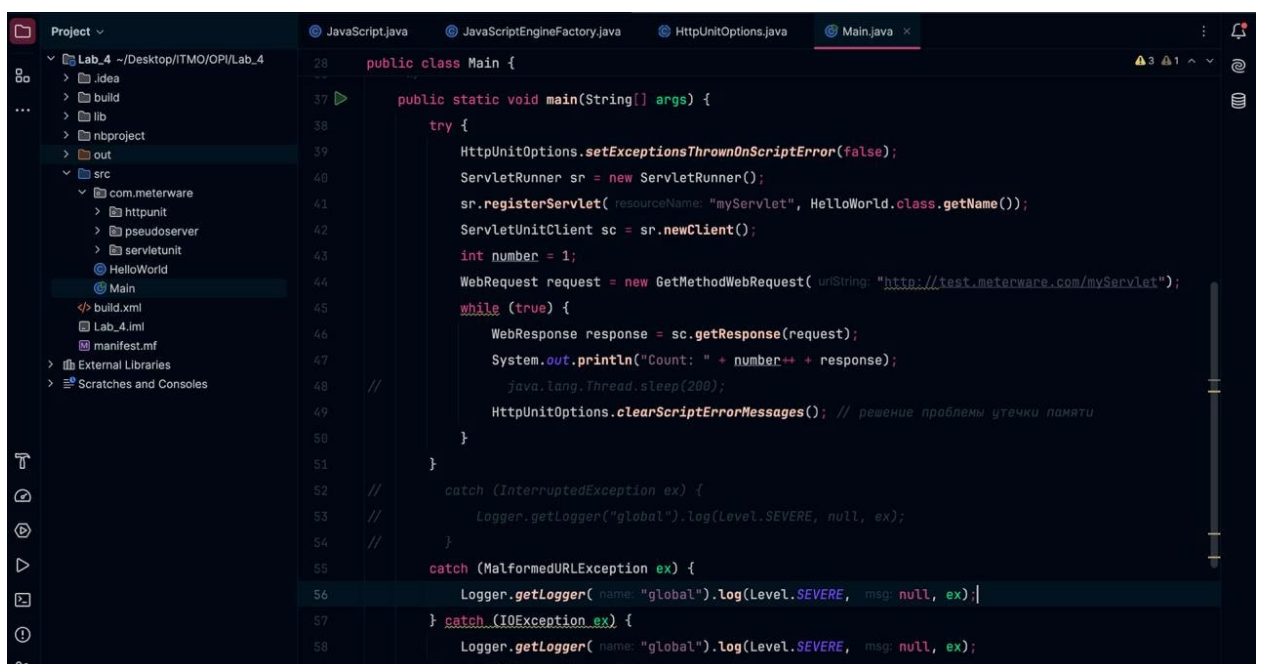
Проанализировав эти строки, заметим, что повторяющиеся строки содержатся в `ArrayList _errorMessagees`. Если углубимся в код, то заметим этот массив сообщений об ошибках и методы для его очистки `clearErrorMessage()`:



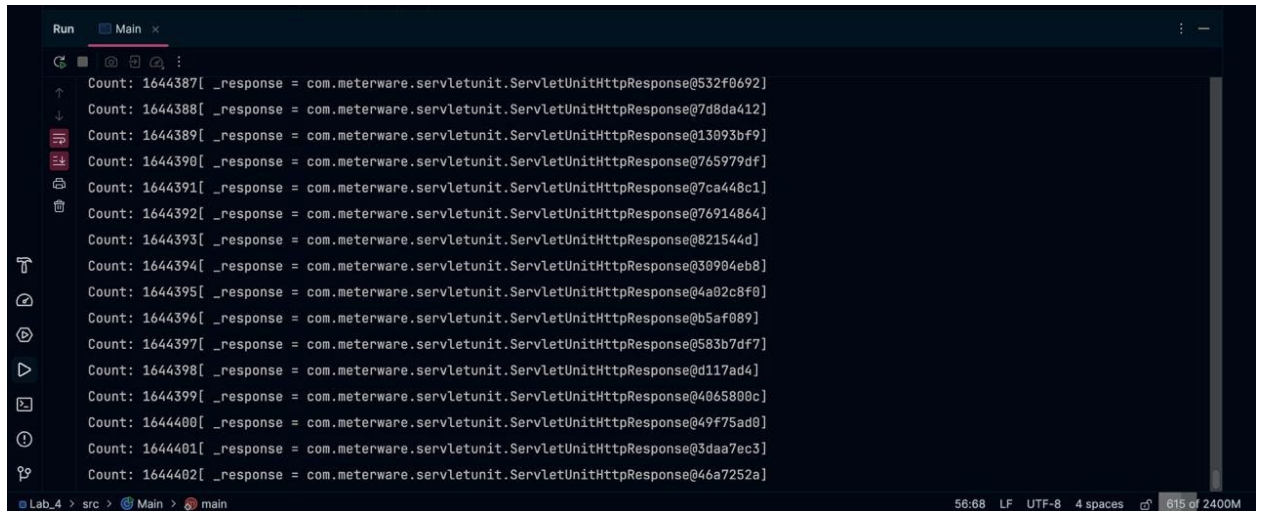
Но если углубимся, заметим, что метод который отвечает за очистку (*clearScriptErrorMessages()*), вообще не используется:



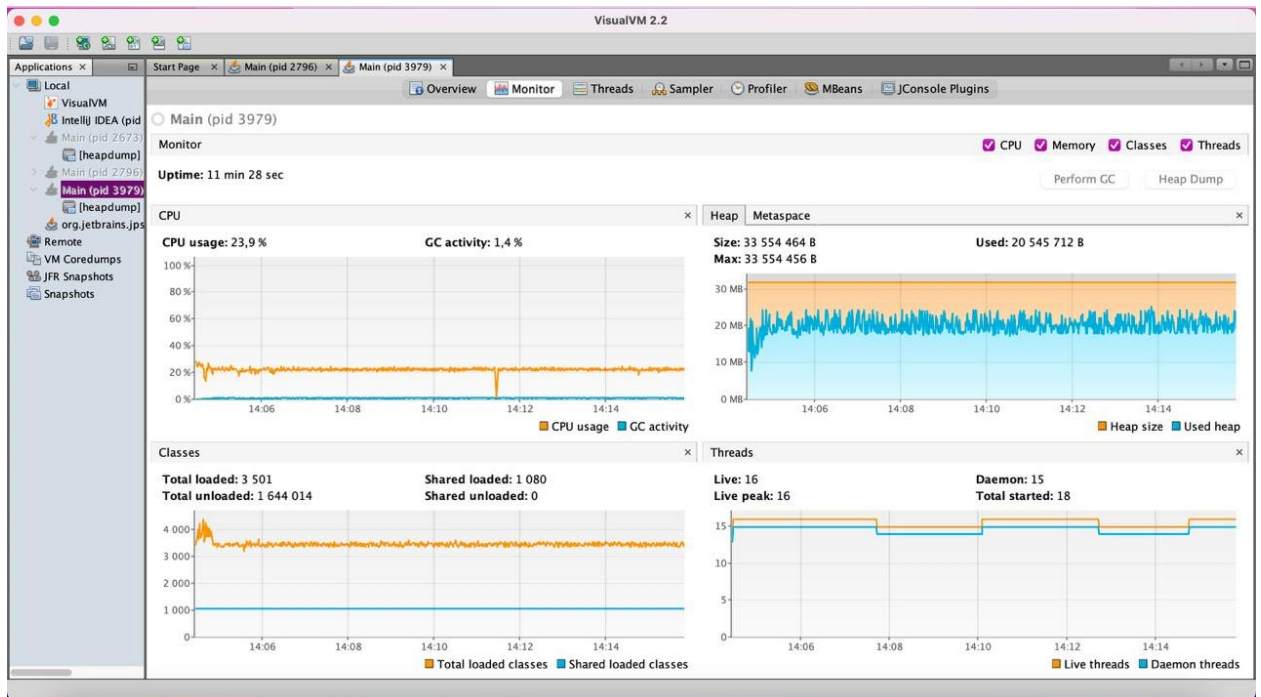
Проанализировав ситуацию, мы можем решить проблему утечки памяти, используя уже реализованный метод `clearScriptErrorMessages()` после выполнения каждого запроса:



Запустив программу, мы можем увидеть, что проблема утечки памяти решена, и теперь программа работает стабильно и не падает с ошибкой `OutOfMemoryException`:



```
Count: 1644387[ _response = com.meterware.servletunit.ServletUnitHttpResponse@532f0692]
Count: 1644388[ _response = com.meterware.servletunit.ServletUnitHttpResponse@7d8da412]
Count: 1644389[ _response = com.meterware.servletunit.ServletUnitHttpResponse@13093bf9]
Count: 1644390[ _response = com.meterware.servletunit.ServletUnitHttpResponse@765979df]
Count: 1644391[ _response = com.meterware.servletunit.ServletUnitHttpResponse@7ca448c1]
Count: 1644392[ _response = com.meterware.servletunit.ServletUnitHttpResponse@76914864]
Count: 1644393[ _response = com.meterware.servletunit.ServletUnitHttpResponse@821544d]
Count: 1644394[ _response = com.meterware.servletunit.ServletUnitHttpResponse@30904eb8]
Count: 1644395[ _response = com.meterware.servletunit.ServletUnitHttpResponse@4a02c8f0]
Count: 1644396[ _response = com.meterware.servletunit.ServletUnitHttpResponse@b5af089]
Count: 1644397[ _response = com.meterware.servletunit.ServletUnitHttpResponse@583b7df7]
Count: 1644398[ _response = com.meterware.servletunit.ServletUnitHttpResponse@d117ad4]
Count: 1644399[ _response = com.meterware.servletunit.ServletUnitHttpResponse@4065800c]
Count: 1644400[ _response = com.meterware.servletunit.ServletUnitHttpResponse@49f75ad0]
Count: 1644401[ _response = com.meterware.servletunit.ServletUnitHttpResponse@3daa7ec3]
Count: 1644402[ _response = com.meterware.servletunit.ServletUnitHttpResponse@46a7252a]
```



6. Вывод

Во время выполнения лабораторной работы мы познакомились 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 и устранять утечки памяти.