Национальный исследовательский университет ИТМО

Факультет ПИиКТ

Лабораторная работа по Методам и средствам программной инженерии №4

Вариант 1994

Работу выполнили:

Асташин С. С., Бадмаев Н. И.

Группа:

P3230

Преподаватель:

Каюков И. А.

Санкт-Петербург, 2022

Задание:

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 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 версию Java Language Specification, реализуемую данной средой исполнения.

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 потока, потребляющего наибольший процент времени CPU.

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:

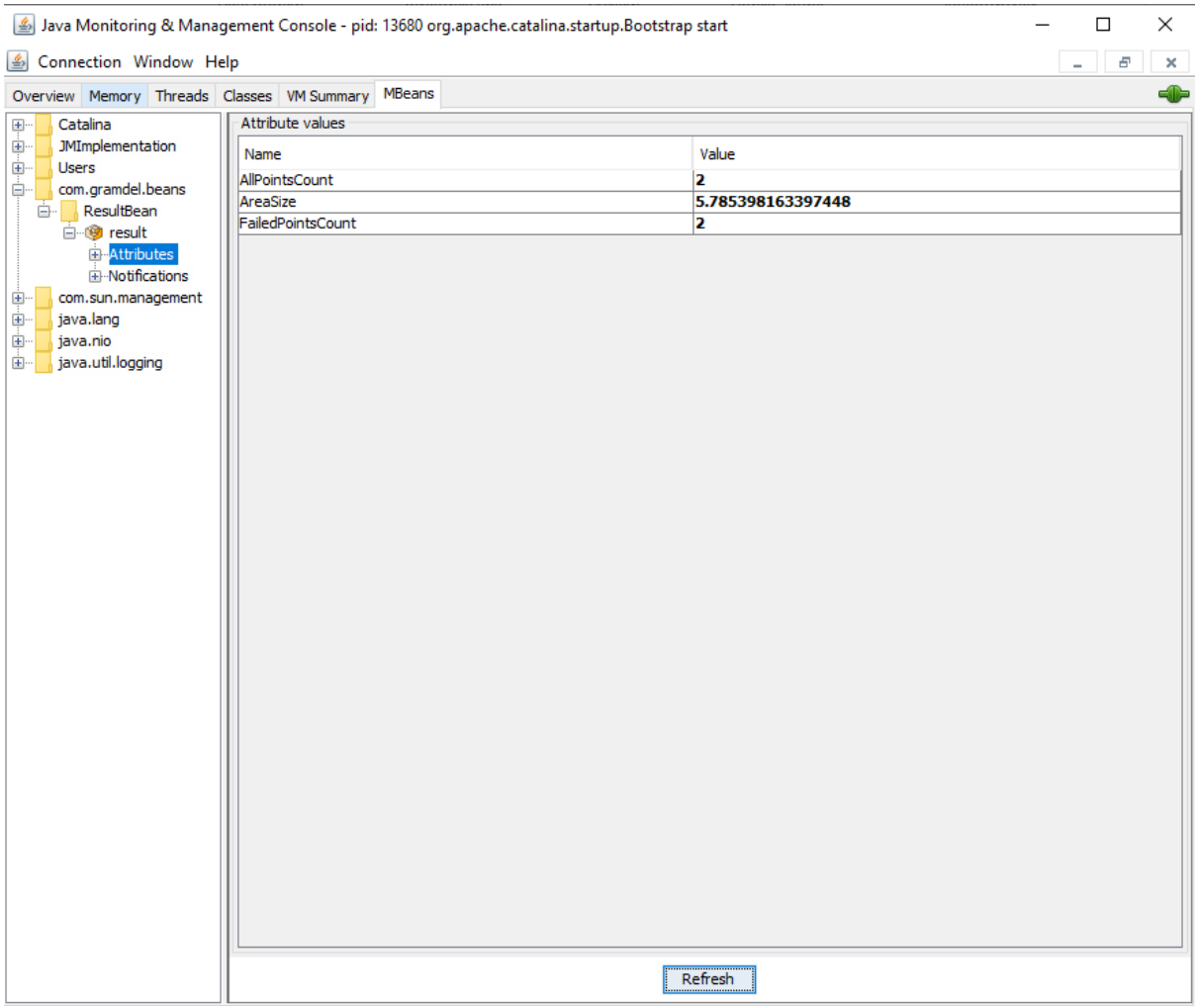
- Описание выявленной проблемы.
- Описание путей устранения выявленной проблемы.
- Подробное (со скриншотами) описание алгоритма действий, который позволил выявить и локализовать проблему.

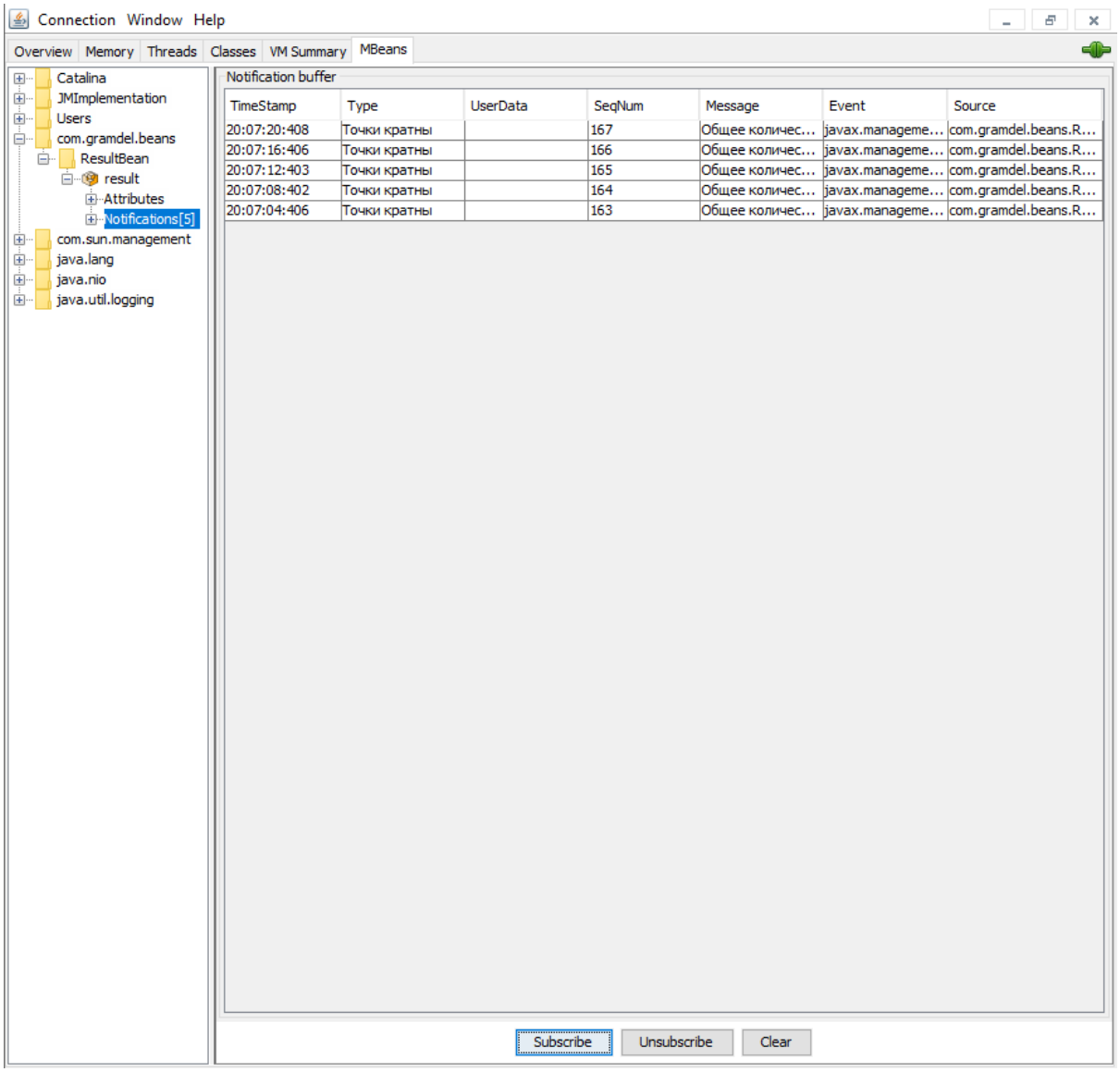
Студент должен обеспечить возможность воспроизведения процесса поиска и локализации проблемы по требованию преподавателя.

Исходный код:

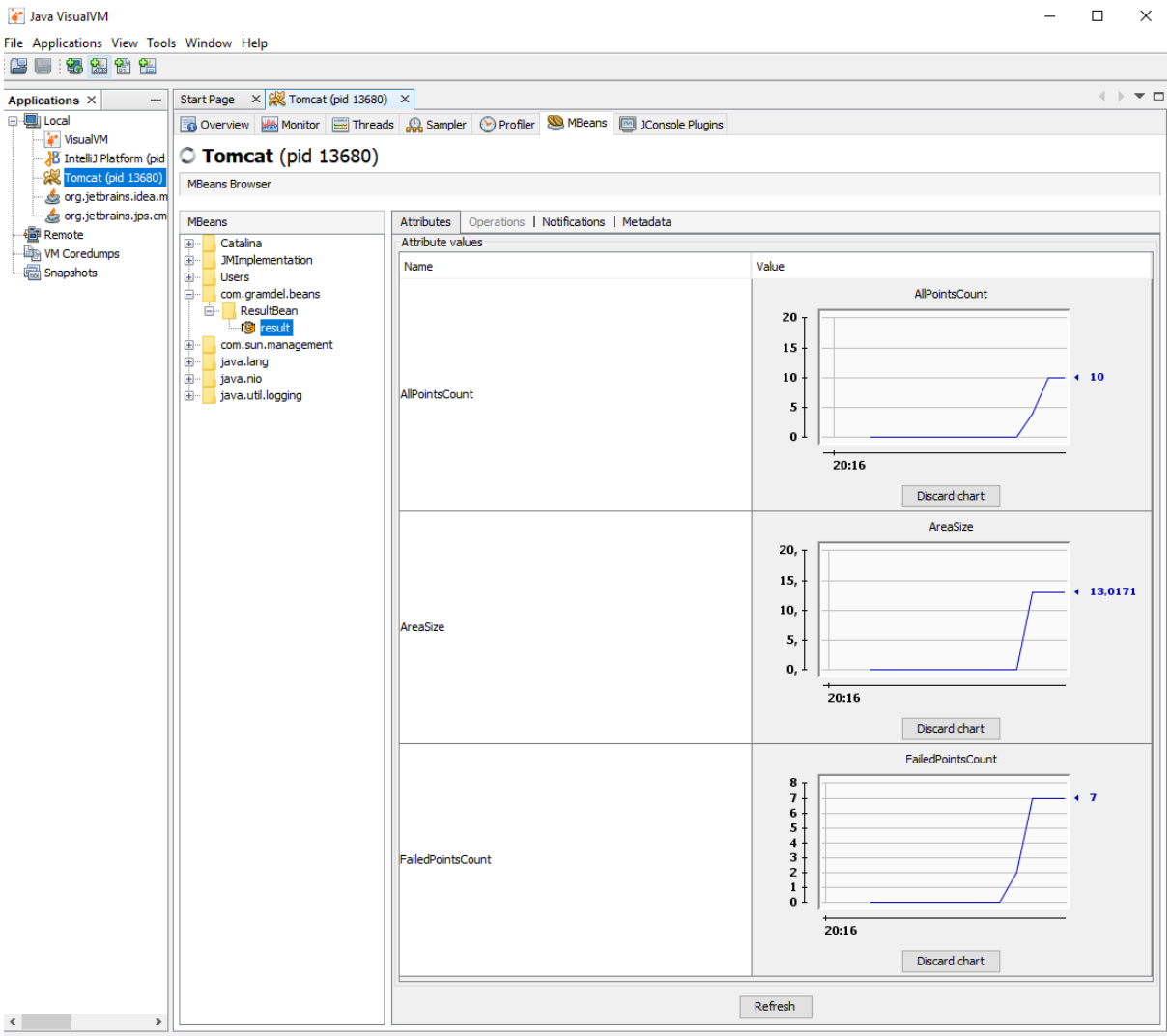
https://github.com/Gramdel/mispi_lab4

JConsole:



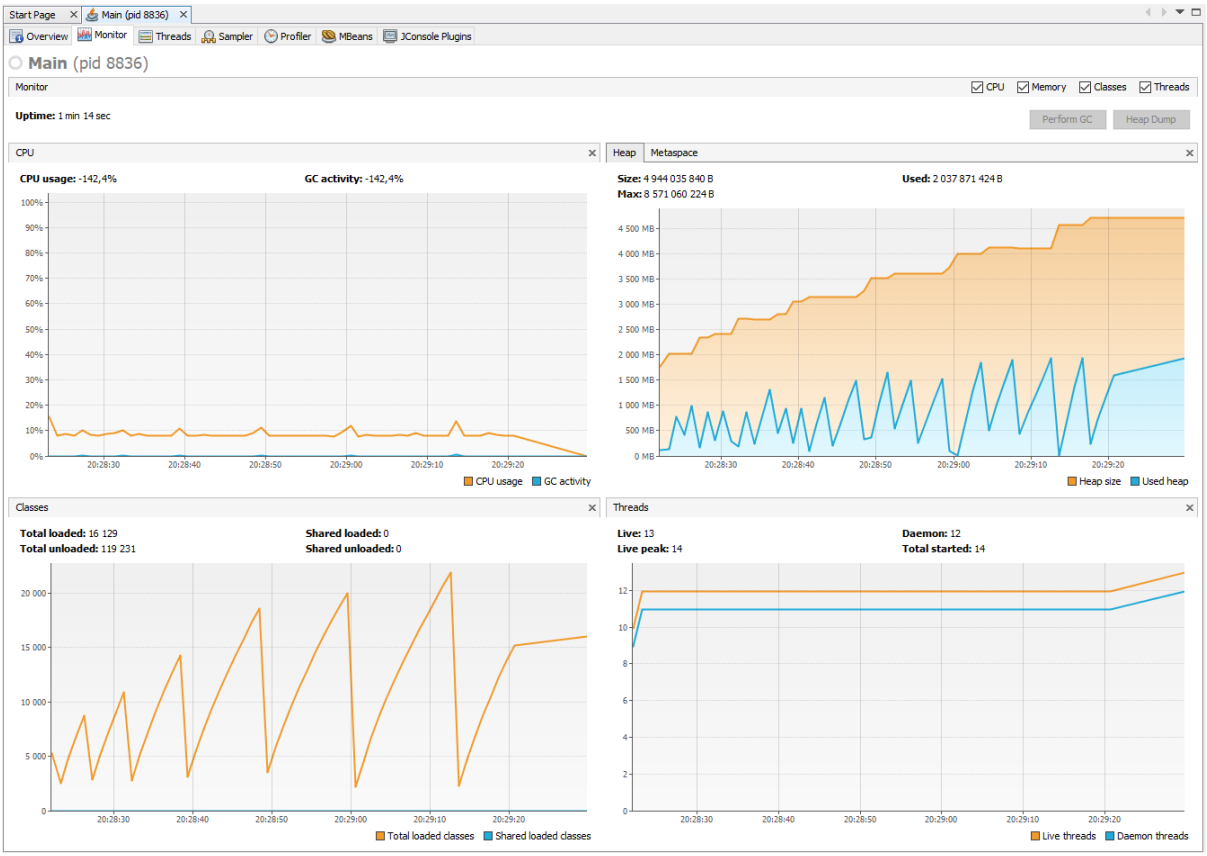

Connection Window Help
Overview Memory Threads Classes VM Summary MBeans
Catalina
JMImplementation
Users
com.gramdel.beans
ResultBean
result
Attributes
Notifications[5]
com.sun.management
java.lang
java.nio
java.util.logging
Notification buffer
TimeStamp Type UserData SeqNum Message Event Source
20:07:20:408 Точки кратны 167 Общее количес... javax.manageme... com.gramdel.beans.R...
20:07:16:406 Точки кратны 166 Общее количес... javax.manageme... com.gramdel.beans.R...
20:07:12:403 Точки кратны 165 Общее количес... javax.manageme... com.gramdel.beans.R...
20:07:08:402 Точки кратны 164 Общее количес... javax.manageme... com.gramdel.beans.R...
20:07:04:406 Точки кратны 163 Общее количес... javax.manageme... com.gramdel.beans.R...
Subscribe
Unsubscribe
Clear

VisualVM:


Java VisualVM
File Applications View Tools Window Help
Applications
Local
VisualVM
IntelliJ Platform (pid
Tomcat (pid 13680)
org.jetbrains.idea.m
org.jetbrains.jps.cm
Remote
VM Coredumps
Snapshots
Start Page
Tomcat (pid 13680)
Overview
Monitor
Threads
Sampler
Profiler
MBeans
JConsole Plugins
Tomcat (pid 13680)
MBeans Browser
MBeans
Catalina
JMImplementation
Users
com.gramdel.beans
ResultBean
result
com.sun.management
java.lang
java.nio
java.util.logging
Attributes
Operations
Notifications
Metadata
Attribute values
Name
Value
AllPointsCount
20
15
10
5
0
20:16
10
Discard chart
AreaSize
AreaSize
20,
15,
10,
5,
0,
20:16
13,0171
Discard chart
FailedPointsCount
FailedPointsCount
8
7
6
5
4
3
2
1
0
20:16
7
Discard chart
Refresh

Есть ли проблемы с производительностью?

Понаблюдаем за поведением программы с помощью вкладки Monitor в VisualVM.



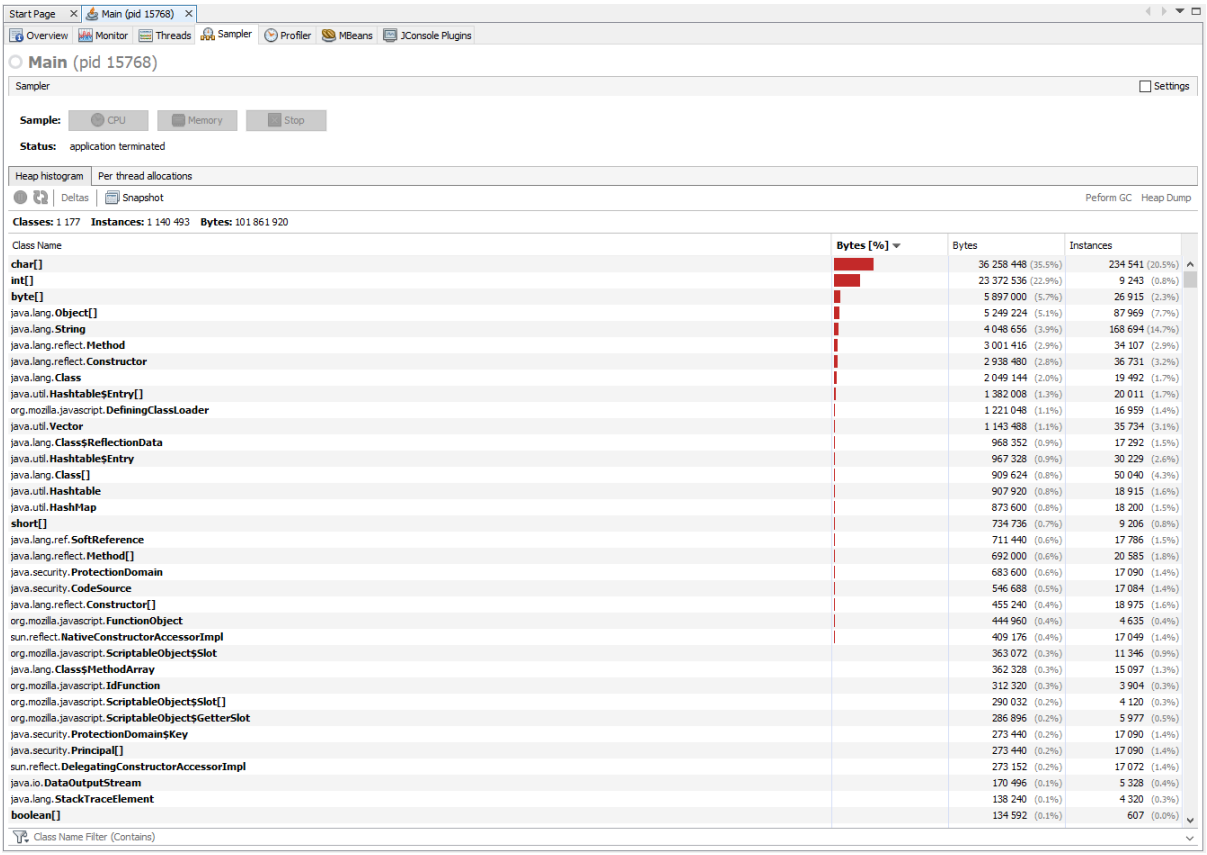
Количество классов постоянно изменяется, причем оно становится всё больше и больше на каждом “пике”, даже несмотря на работу GC. Количество отработанных классов тоже растёт.

Также, по графику использования кучи видно, что к ней постоянно происходит обращение, её размер постоянно меняется. Это негативно сказывается на скорости работы программы, поскольку происходят постоянные вызовы GC.

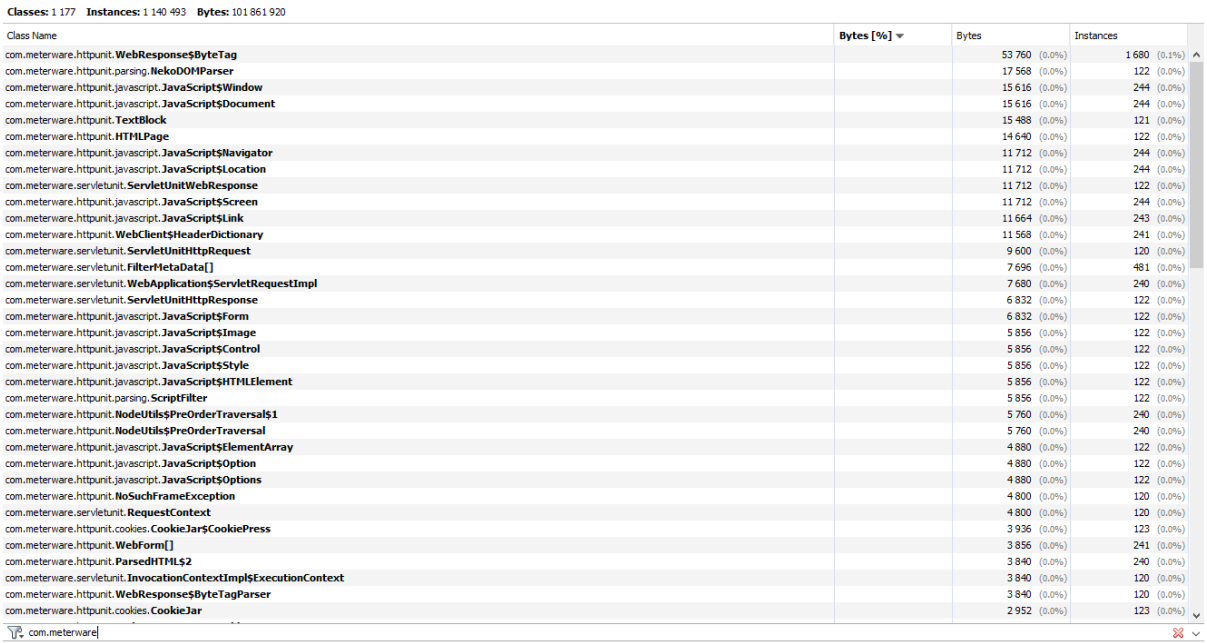
Таким образом, проблемы однозначно есть.

Поиск виновников:

Перейдём во вкладку Sampler и посмотрим на память.



Как можно заметить, количество массивов char’ов постоянно растёт, несмотря на работу GC. Найдём, где это происходит (в каком классе пакета нашей программы, com.meterware):



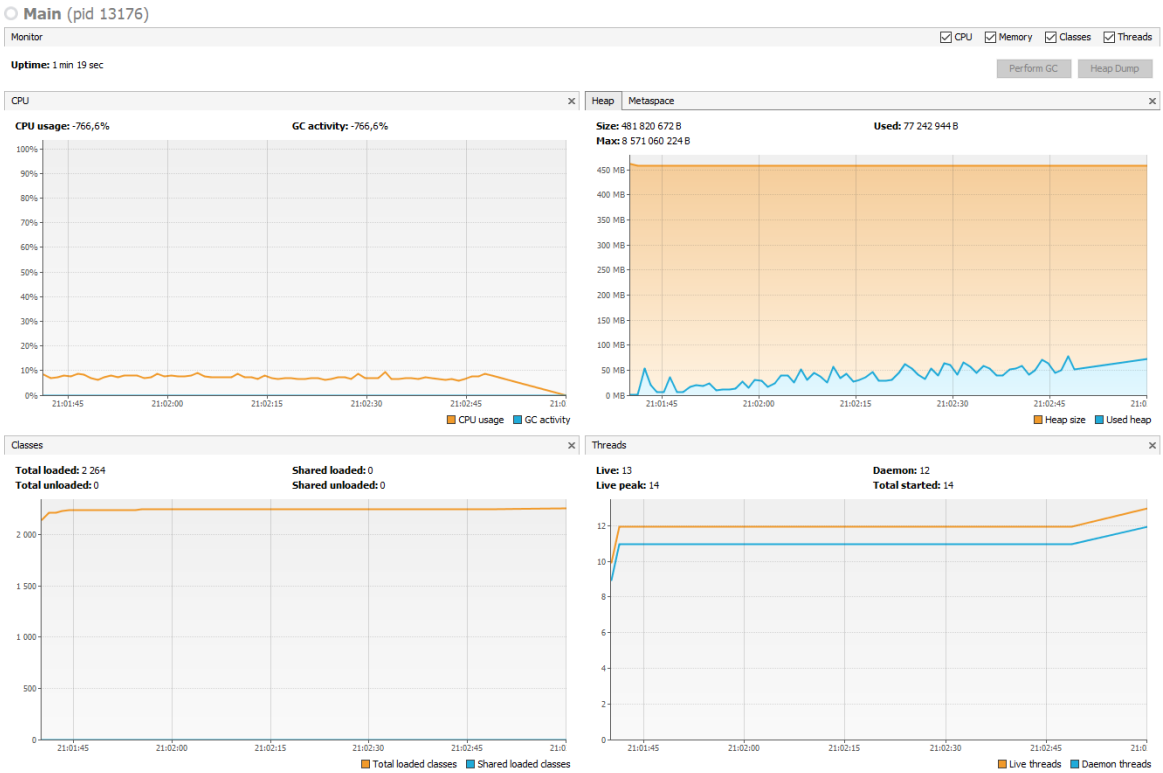
Судя по всему, дело в “WebResponse”. Исходный код метода main в студию!

```
HttpUnitOptions.setExceptionsThrownOnScriptError(false);
ServletRunner sr = new ServletRunner();
sr.registerServlet( resourceName: "myServlet", HelloWorld.class.getName());
ServletUnitClient sc = sr.newClient();
int number = 1;
WebRequest request = new GetMethodWebRequest( urlString: "http://test.meterware.com/myServlet");
while (true) {
    WebResponse response = sc.getResponse(request);
    System.out.println("Count: " + number++ + response);
    java.lang.Thread.sleep( millis: 200);
}
```

Тут невооруженным взглядом видно, что в бесконечном цикле постоянно создается объект WebResponse, но передаётся ему один и тот же объект WebRequest, который создан до цикла, а значит и ответ будет всегда один и тот же. Таким образом, можно просто вынести WebResponse за пределы цикла:

```
WebRequest request = new GetMethodWebRequest( urlString: "http://test.meterware.com/myServlet");
WebResponse response = sc.getResponse(request);
while (true) {
    System.out.println("Count: " + number++ + response);
    java.lang.Thread.sleep( millis: 200);
}
```

Проверим, ушла ли проблема:



Всё отлично, количество классов больше не растёт, куча не растёт, графики более плавные, обращений к GC тоже меньше.

Выводы:

В результате данной работы были изучены JMX и MBean'ы; произошло знакомство с JConsole и VisualVM. С помощью последнего были выявлены и устранены проблемы в представленной программе (кстати, можно ещё сделать вывод, что иногда всё может пойти не так из-за одной строчки кода :).