



УНИВЕРСИТЕТ ИТМО

Лабораторная работа №4

Дисциплина: Методы и средства программной инженерии

Преподаватель: Клименков С. В., Цопа Е. А.

Факультет: ПИиКТ

Авторы: Бусыгин Иван, Залевский Дмитрий

Группа: Р3212.

Вариант: 1236

Санкт-Петербург
2022 год

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, попадающих в область. В случае, если пользователь совершил 4 "промаха" подряд, 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 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.

Отчёт по работе должен содержать:

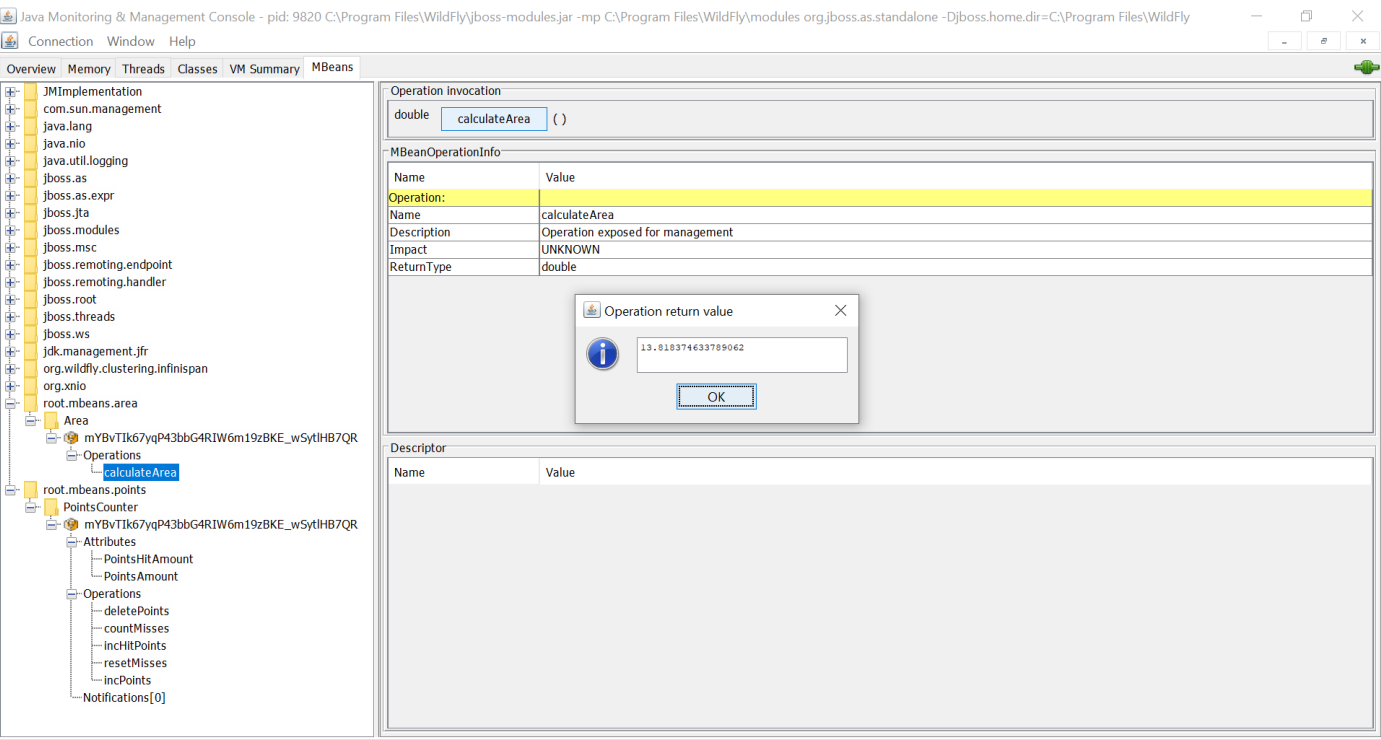
1. Текст задания.
2. Исходный код разработанных MBean-классов и сопутствующих классов.
3. Скриншоты программы JConsole со снятыми показаниями, выводы по результатам мониторинга.
4. Скриншоты программы VisualVM со снятыми показаниями, выводы по результатам профилирования.
5. Скриншоты программы VisualVM с комментариями по ходу поиска утечки памяти.
6. Выводы по работе.

Мониторинг и профилирование веб-приложения.

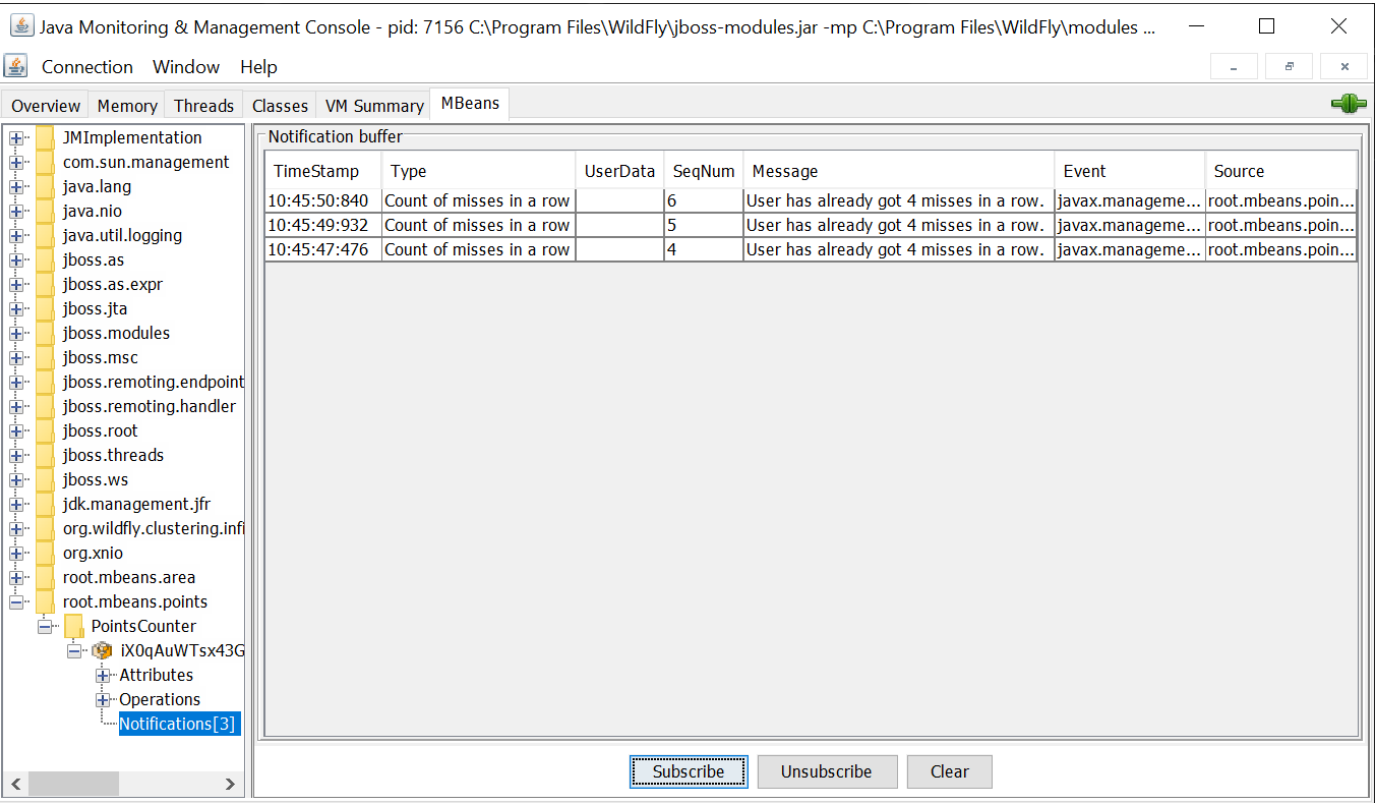
Код программы, включая М-бины можно посмотреть в репозитории на GitHub:
https://github.com/BusyginIvan/second_course/tree/master/software_engineering/lab4

Мониторинг в JConsole.

Вычисление площади фигуры:



Получение уведомлений о четырёх “промахах” подряд:



Во вкладке VM Summary мы можем видеть, что среда выполнения реализует версию 18.0.1.1 спецификации Java:

Java Monitoring & Management Console - pid: 7156 C:\Program Files\WildFly\jboss-modules.jar -mp C:\Program Files\WildFly\modules org.jboss.as.standalone -Djboss.home.dir=C:\Program Files\WildFly

Connection Window Help

Overview Memory Threads Classes VM Summary MBeans

VM Summary

Friday, June 3, 2022 at 10:51:37 AM Moscow Standard Time

Connection name: pid: 7156 C:\Program Files\WildFly\jboss-modules.jar -mp C:\Program Files\WildFly\modules org.jboss.as.standalone -Djboss.home.dir=C:\Program Files\WildFly

Uptime: 6 minutes

Virtual Machine: Java HotSpot(TM) 64-Bit Server VM version 18.0.1.1+2-6

Process CPU time: 40.921 seconds

Vendor: Oracle Corporation

JIT compiler: HotSpot 64-Bit Tiered Compilers

Name: 7156@LAPTOP-47SHSMGO

Total compile time: 28.722 seconds

Live threads: 59

Current classes loaded: 19,509

Peak: 146

Total classes loaded: 19,509

Daemon threads: 26

Total classes unloaded: 0

Total threads started: 156

Current heap size: 110,300 kbytes

Committed memory: 264,192 kbytes

Maximum heap size: 524,288 kbytes

Pending finalization: 0 objects

Garbage collector: Name = 'G1 Young Generation', Collections = 29, Total time spent = 0.184 seconds

Garbage collector: Name = 'G1 Old Generation', Collections = 0, Total time spent = 0.000 seconds

Operating System: Windows 10 10.0

Total physical memory: 20,848,884 kbytes

Architecture: amd64

Free physical memory: 9,195,000 kbytes

Number of processors: 4

Total swap space: 23,994,612 kbytes

Committed virtual memory: 564,436 kbytes

Free swap space: 9,723,748 kbytes

VM arguments: -Dprogram.name=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.desktop/sun.awt=ALL-UNNAMED --add-exports=java.naming/com.sun.jndi ldap=ALL-UNNAMED --add-opens=java.base/java.lang=ALL-UNNAMED --add-opens=java.base/java.lang.invoke=ALL-UNNAMED --add-opens=java.base/java.io=ALL-UNNAMED --add-opens=java.base/java.lang.reflect=ALL-UNNAMED --add-opens=java.base/java.security=ALL-UNNAMED --add-opens=java.base/java.util=ALL-UNNAMED --add-opens=java.base/java.util.concurrent=ALL-UNNAMED --add-opens=java.management/javax.management=ALL-UNNAMED --add-opens=java.naming/javax.naming=ALL-UNNAMED -Dorg.jboss.boot.log.file=C:\Program Files\WildFly\standalone\log\server.log -Dlogging.configuration=file:C:\Program Files\WildFly\standalone\configuration\logging.properties

Мониторинг и профилирование в VisualVM.

Графики изменения показаний MBean-классов:

VisualVM 2.1.3

File Applications View Tools Window Help

Applications ×

Local

IntelliJ IDEA (pid 2080)

VisualVM

org.jboss.modules.Main

Remote

VM CoreDumps

JFR Snapshots

Snapshots

org.jboss.modules.Main (pid 16332) ×

Overview Monitor Threads Sampler Profiler MBeans

org.jboss.modules.Main (pid 16332)

MBeans Browser

MBeans

com.sun.management

java.lang

java.nio

java.util.logging

jboss.as

jboss.as.expr

jboss.jta

jboss.modules

jboss.msc

jboss.remoting.endpoint

jboss.remoting.handler

jboss.root

jboss.threads

jboss.ws

jdk.management.jfr

org.wildfly.clustering.infinispan

org.xmlio

root.mbeans.area

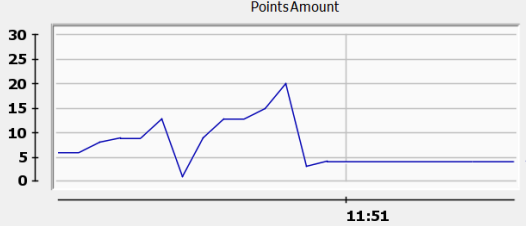
root.mbeans.points

PointsCounter

6JXkQOWK-qaD-yNzGd

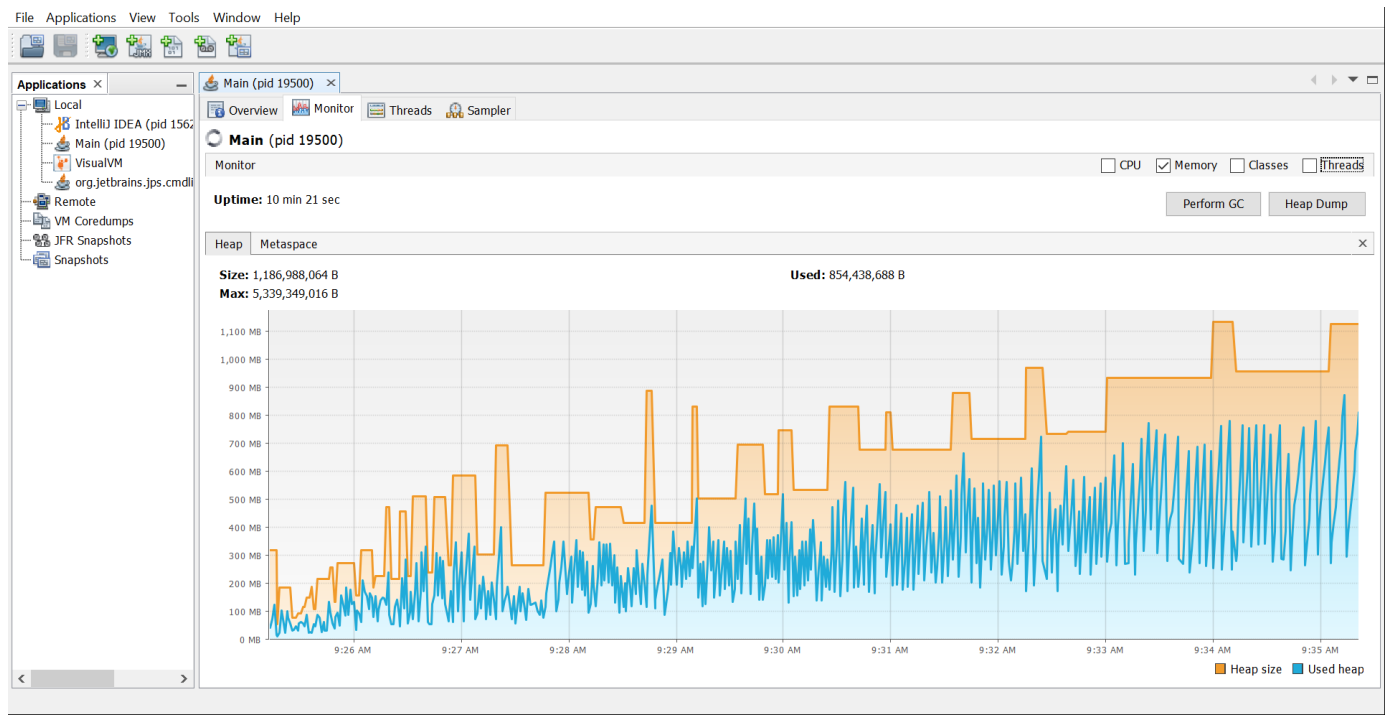
Attributes Operations Notifications Metadata

Attribute values

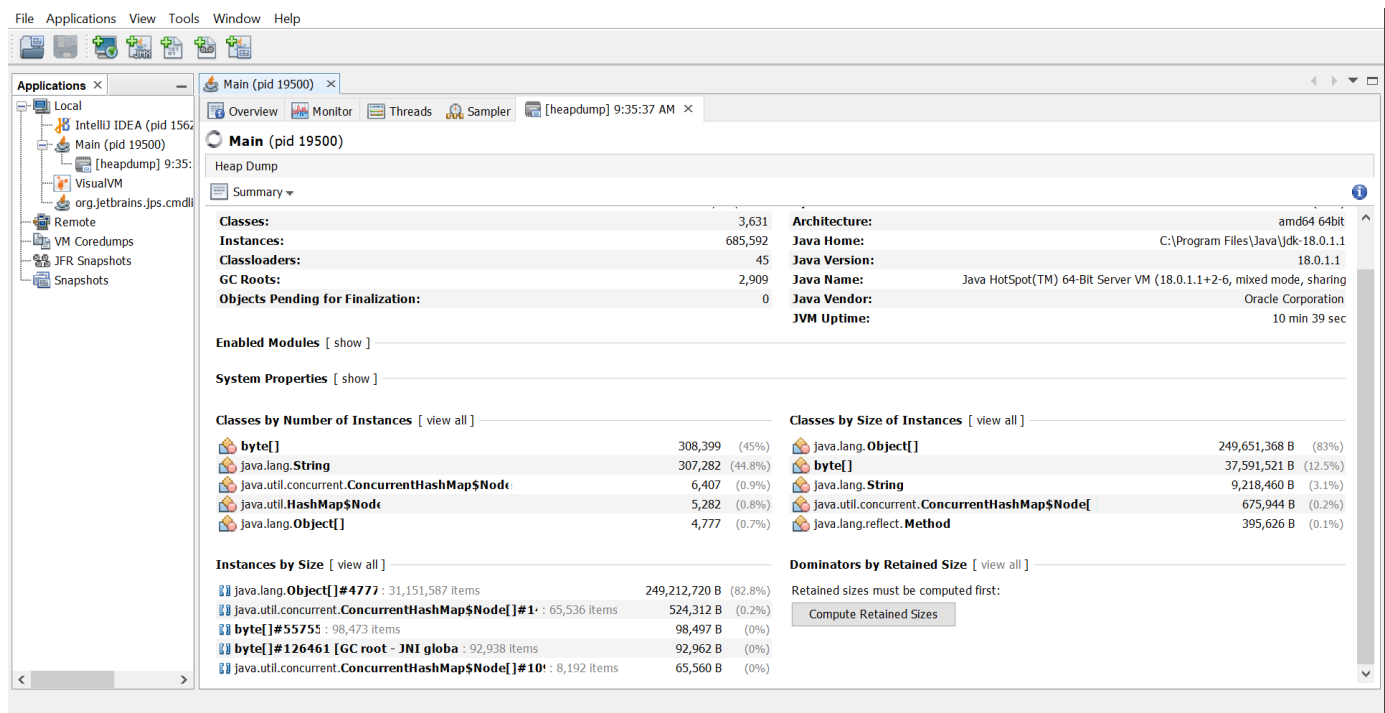
Name	Value
PointsAmount	<div>PointsAmount</div> <div></div> <div>11:51</div> <div>Discard chart</div>
PointsHitAmount	3

Поиск и устранение утечки памяти.

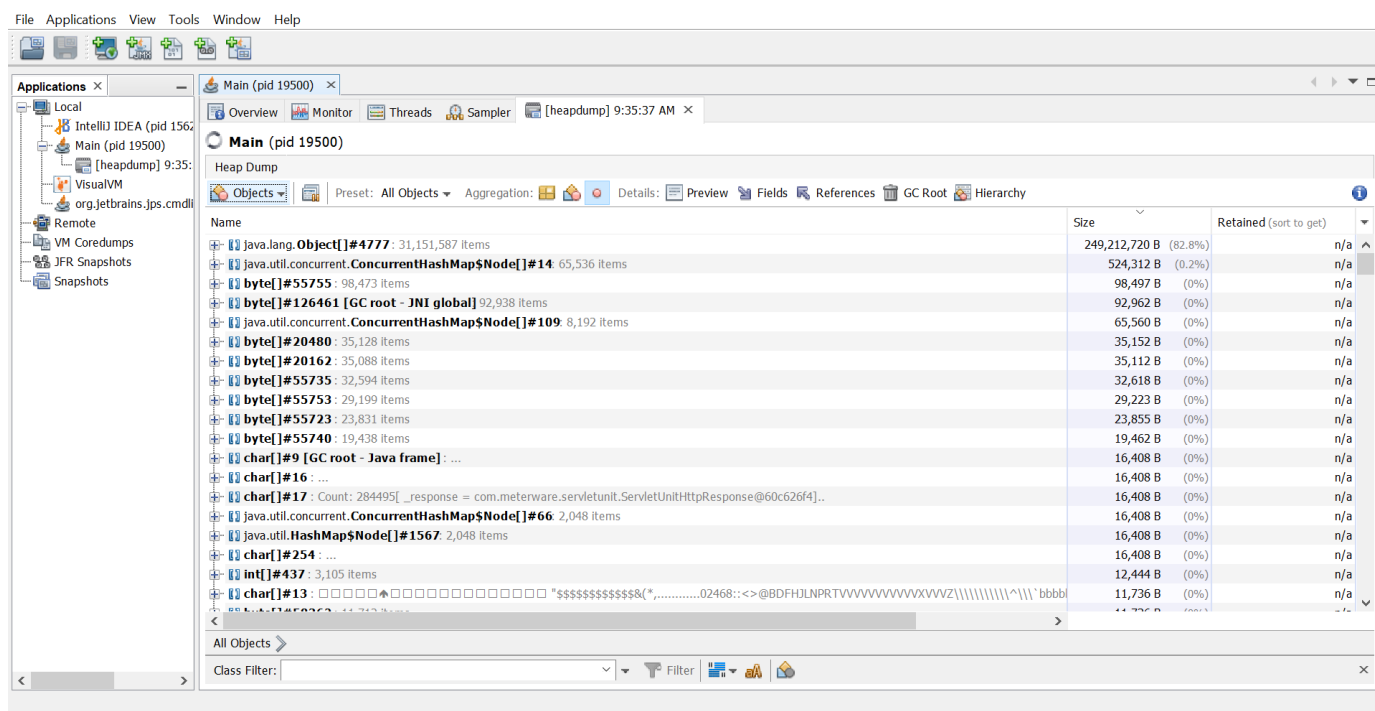
Запускаем приложение с VisualVM. Видим, что неуклонно растёт потребление памяти приложением. Возможно, это утечка.



Делаем Heap Dump.

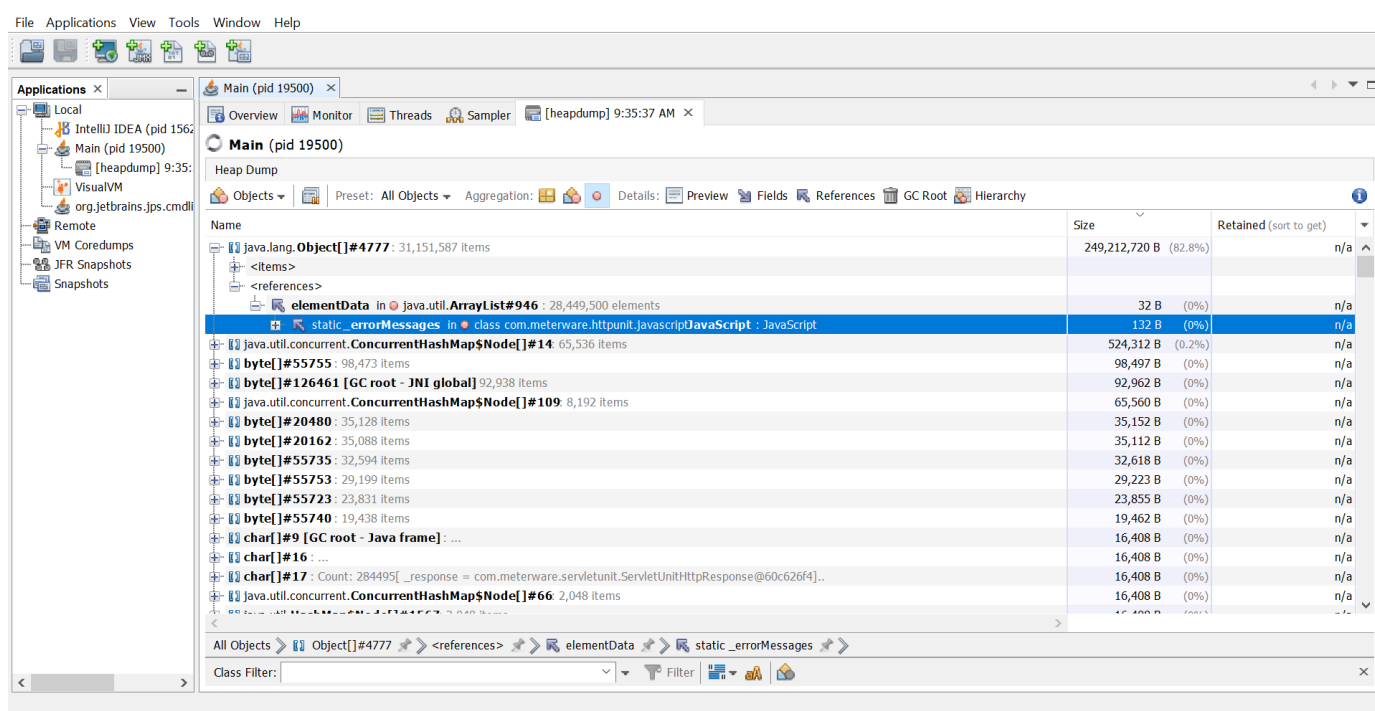


Переходим в раздел, где объекты отсортированы по размеру потребляемой памяти. Видим, что большую часть памяти занимает один массив объектов.



Name	Size	Retained (sort to get)
java.lang.Object[]#4777 : 31,151,587 items	249,212,720 B (82.8%)	n/a
java.util.concurrent.ConcurrentHashMap\$Node[#14 : 65,536 items	524,312 B (0.2%)	n/a
byte[]#55755 : 98,473 items	98,497 B (0%)	n/a
byte[]#126461 [GC root - JNI global] : 92,938 items	92,962 B (0%)	n/a
java.util.concurrent.ConcurrentHashMap\$Node[#109 : 8,192 items	65,560 B (0%)	n/a
byte[]#20480 : 35,128 items	35,152 B (0%)	n/a
byte[]#20162 : 35,088 items	35,112 B (0%)	n/a
byte[]#55735 : 32,594 items	32,618 B (0%)	n/a
byte[]#55753 : 29,199 items	29,223 B (0%)	n/a
byte[]#55723 : 23,831 items	23,855 B (0%)	n/a
byte[]#55740 : 19,438 items	19,462 B (0%)	n/a
char[]#9 [GC root - Java frame] : ...	16,408 B (0%)	n/a
char[]#16 : ...	16,408 B (0%)	n/a
Count: 284495[_response = com.meterware.servletunit.ServletUnitHttpResponse@60c626f4]..	16,408 B (0%)	n/a
java.util.concurrent.ConcurrentHashMap\$Node[#66 : 2,048 items	16,408 B (0%)	n/a
java.util.HashMap\$Node[#1567 : 2,048 items	16,408 B (0%)	n/a
char[]#254 : ...	16,408 B (0%)	n/a
int[]#437 : 3,105 items	12,444 B (0%)	n/a
char[]#13 : Count: 02468::<: @BDFHJLNPRTVWWWWWVXVWVZ '`' bbb	11,736 B (0%)	n/a

Разворачиваем информацию о родительских объектах массива. Видим, что это ArrayList в классе JavaScript.



Name	Size	Retained (sort to get)
java.lang.Object[]#4777 : 31,151,587 items	249,212,720 B (82.8%)	n/a
<items>		
<references>		
elementData in java.util.ArrayList#946 : 28,449,500 elements	32 B (0%)	n/a
static_errorMessages in class com.meterware.httpunit.JavaScript : JavaScript	132 B (0%)	n/a
java.util.concurrent.ConcurrentHashMap\$Node[#14 : 65,536 items	524,312 B (0.2%)	n/a
byte[]#55755 : 98,473 items	98,497 B (0%)	n/a
byte[]#126461 [GC root - JNI global] : 92,938 items	92,962 B (0%)	n/a
java.util.concurrent.ConcurrentHashMap\$Node[#109 : 8,192 items	65,560 B (0%)	n/a
byte[]#20480 : 35,128 items	35,152 B (0%)	n/a
byte[]#20162 : 35,088 items	35,112 B (0%)	n/a
byte[]#55735 : 32,594 items	32,618 B (0%)	n/a
byte[]#55753 : 29,199 items	29,223 B (0%)	n/a
byte[]#55723 : 23,831 items	23,855 B (0%)	n/a
byte[]#55740 : 19,438 items	19,462 B (0%)	n/a
char[]#9 [GC root - Java frame] : ...	16,408 B (0%)	n/a
char[]#16 : ...	16,408 B (0%)	n/a
Count: 284495[_response = com.meterware.servletunit.ServletUnitHttpResponse@60c626f4]..	16,408 B (0%)	n/a
java.util.concurrent.ConcurrentHashMap\$Node[#66 : 2,048 items	16,408 B (0%)	n/a

Исправляем ошибку в коде.

Смотрим код и видим, что в классе JavaScript в статический List добавляются сообщения об ошибках. При этом приложение никак не контролирует рост этого списка и не чистит его, хоть в классе и есть метод clearErrorMessages.

Решить проблему можно путём сохранения сообщений об ошибках в файл. Так ошибки не будут занимать место в куче, и администратор сможет при необходимости вручную удалить разросшийся файл логов. Новая реализация методов обработки ошибок представлена ниже:

```
private static final File ERR_LOGS_FILE = new File("errLogs.txt");

static void clearErrorMessages() { ERR_LOGS_FILE.delete(); }
```



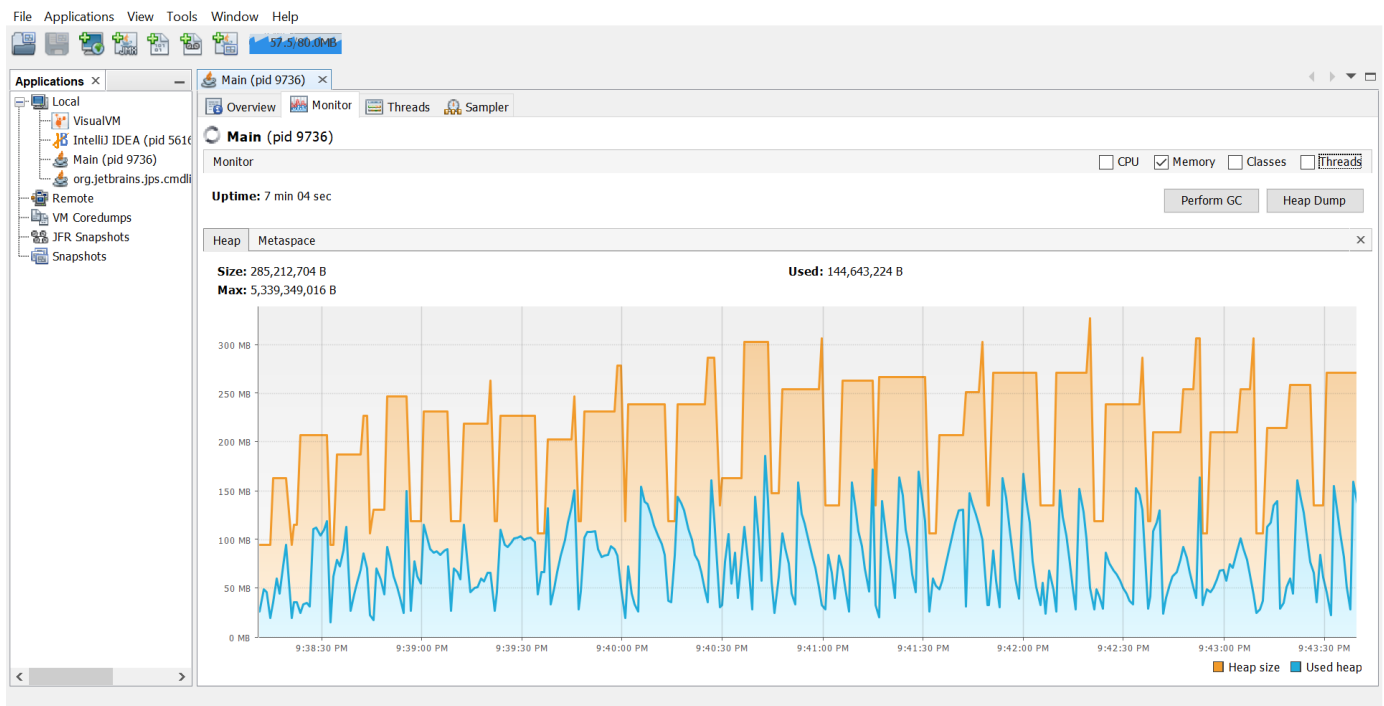
```

static String[] getErrorMessages() {
    try (FileReader fileReader = new FileReader(ERR_LOGS_FILE)) {
        try (BufferedReader bufferedReader = new BufferedReader(fileReader)) {
            return (String[]) bufferedReader.lines().toArray();
        }
    } catch (FileNotFoundException e) {
        return null;
    } catch (IOException e) {
        System.err.println(e.getMessage()); return null;
    }
}

static void addErrorMessage(String message) {
    try (FileWriter writer = new FileWriter(ERR_LOGS_FILE, true)) {
        writer.write(message + '\n');
        writer.flush();
    } catch (IOException e) {
        System.err.println(e.getMessage());
    }
}
}

```

После проделанных изменений в коде снова запускаем приложение с VisualVM. Видим, что размер кучи теперь не растёт. Утечка устранена.



Вывод.

Пусть только попробуют утечки памяти или проблемы с производительностью закрасться в наше ПО. Мы их тут же выведем на чистую воду и поступим с ними так, как они того заслуживают. Мы готовы дать достойный отпор любым неприятностям, проявляющимся на этапе тестирования производительности.