Федеральное государственное автономное образовательное учреждение высшего образования **«Национальный исследовательский университет ИТМО»**

Факультет Программной Инженерии и Компьютерной Тефтели

Лабораторная работа №4
по дисциплине **«Основы программной инженерии»**

Вариант: **1276**

Преподаватель:
Миху Вадим Дмитриевич

Выполнил: Садовой Григорий
Группа: P3207

Санкт-Петербург, 2025

Оглавление

1. Задание

Вариант: **1276**

Вариант 1276

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, 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 время (в мс), прошедшее с момента запуска виртуальной машины.

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.

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

1. Исходный код разработанных MBean

- AreaCalculator.java

```java
package itmo.web.demo1.beans;

import java.io.Serializable;
import java.util.List;
import itmo.web.demo1.database.models.Point;

public class AreaCalculator implements AreaCalculatorMBean, Serializable {
    private List<Point> hitPoints;

    public void setHitPoints(List<Point> points) { this.hitPoints = points; }

    @Override
    public double getArea() {
        if (hitPoints == null || hitPoints.size() < 3) return 0.0;

        double area = 0;
        for (int i = 0; i < hitPoints.size(); i++) {
            Point p1 = hitPoints.get(i);
            Point p2 = hitPoints.get((i + 1) % hitPoints.size());
            area += (p1.getX() * p2.getY()) - (p2.getX() * p1.getY());
        }
        return Math.abs(area / 2.0);
    }
}
```

-

- AreaCalculator.java

```java
package itmo.web.demo1.beans;

public interface AreaCalculatorMBean {
    double getArea();
}
```

- MBeanRegistry.java

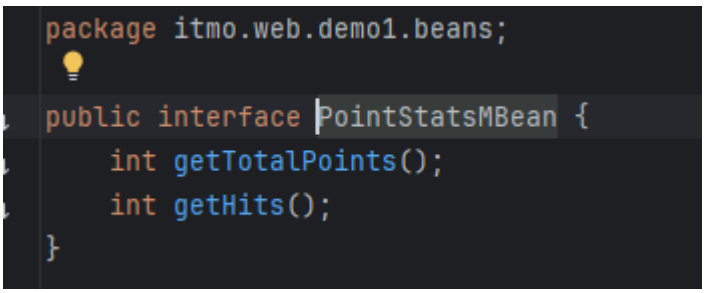
```java
package itmo.web.demo1.beans;

import javax.management.*;
import java.lang.management.ManagementFactory;
import java.util.HashMap;
import java.util.Map;

public class MBeanRegistry {
    private static final Map<Class<?>, ObjectName> beans = new HashMap<>();

    public static void registerBean(Object bean, String name) {
        try {
            String domain = bean.getClass().getPackageName();
            String type = bean.getClass().getSimpleName();
            ObjectName objectName = new ObjectName(String.format("%s:type=%s,name=%s", domain, type, name));

            ManagementFactory.getPlatformMBeanServer().registerMBean(bean, objectName);
            beans.put(bean.getClass(), objectName);
            System.out.println("Registered MBean: " + objectName);
        } catch (InstanceAlreadyExistsException | MBeanRegistrationException | NotCompliantMBeanException |
                 MalformedObjectNameException e) {
            e.printStackTrace();
        }
    }

    public static void unregisterBean(Object bean) {
        try {
            ObjectName name = beans.remove(bean.getClass());
            if (name != null) {
                ManagementFactory.getPlatformMBeanServer().unregisterMBean(name);
                System.out.println("Unregistered MBean: " + name);
            }
        } catch (InstanceNotFoundException | MBeanRegistrationException e) {
            e.printStackTrace();
        }
    }
}
```

- PointStats.java

```java
package itmo.web.demo1.beans;

import javax.management.*;
import java.io.Serializable;
import java.util.concurrent.atomic.AtomicInteger;


public class PointStats implements PointStatsMBean,
        NotificationBroadcaster,
        Serializable {


    private final AtomicInteger totalPoints = new AtomicInteger();
    private final AtomicInteger hits        = new AtomicInteger();


    private final AtomicInteger lastNotifiedMultiple = new AtomicInteger( initialValue: 0);


    private final NotificationBroadcasterSupport broadcaster =
            new NotificationBroadcasterSupport();



    @Override public int getTotalPoints() { return totalPoints.get(); }
    @Override public int getHits()        { return hits.get(); }



    public void initCounters(int total, int hit) {
        totalPoints.set(total);
        hits.set(hit);
        lastNotifiedMultiple.set((total / 10) * 10);
    }
```

```
    public void update(boolean isHit) {
        int total = totalPoints.incrementAndGet();
        if (isHit) hits.incrementAndGet();

        if (total % 10 == 0 &&
                lastNotifiedMultiple.compareAndSet( expectedValue: total - 10, total)) {
            broadcaster.sendNotification(new Notification(
                    type: "point.count.threshold",
                    source: this,
                    System.currentTimeMillis(),
                    message: "User placed " + total + " points (multiple of 10)"));
        }
    }


    public void reset() {
        totalPoints.set(0);
        hits.set(0);
        lastNotifiedMultiple.set(0);
    }



    @Override
    public void addNotificationListener(NotificationListener l,
                                        NotificationFilter f, Object h) {
        broadcaster.addNotificationListener(l, f, h);
    }

    @Override
    public void removeNotificationListener(NotificationListener l)
            throws ListenerNotFoundException {
        broadcaster.removeNotificationListener(l);
    }
        }

        @Override
        public MBeanNotificationInfo[] getNotificationInfo() {
            String[] types = { "point.count.threshold" };
            return new MBeanNotificationInfo[] {
                    new MBeanNotificationInfo(types,
                            Notification.class.getName(),
                            description: "Every time totalPoints becomes a multiple of 10")
            };
        }
    }
```

- PointStatsMBean.java

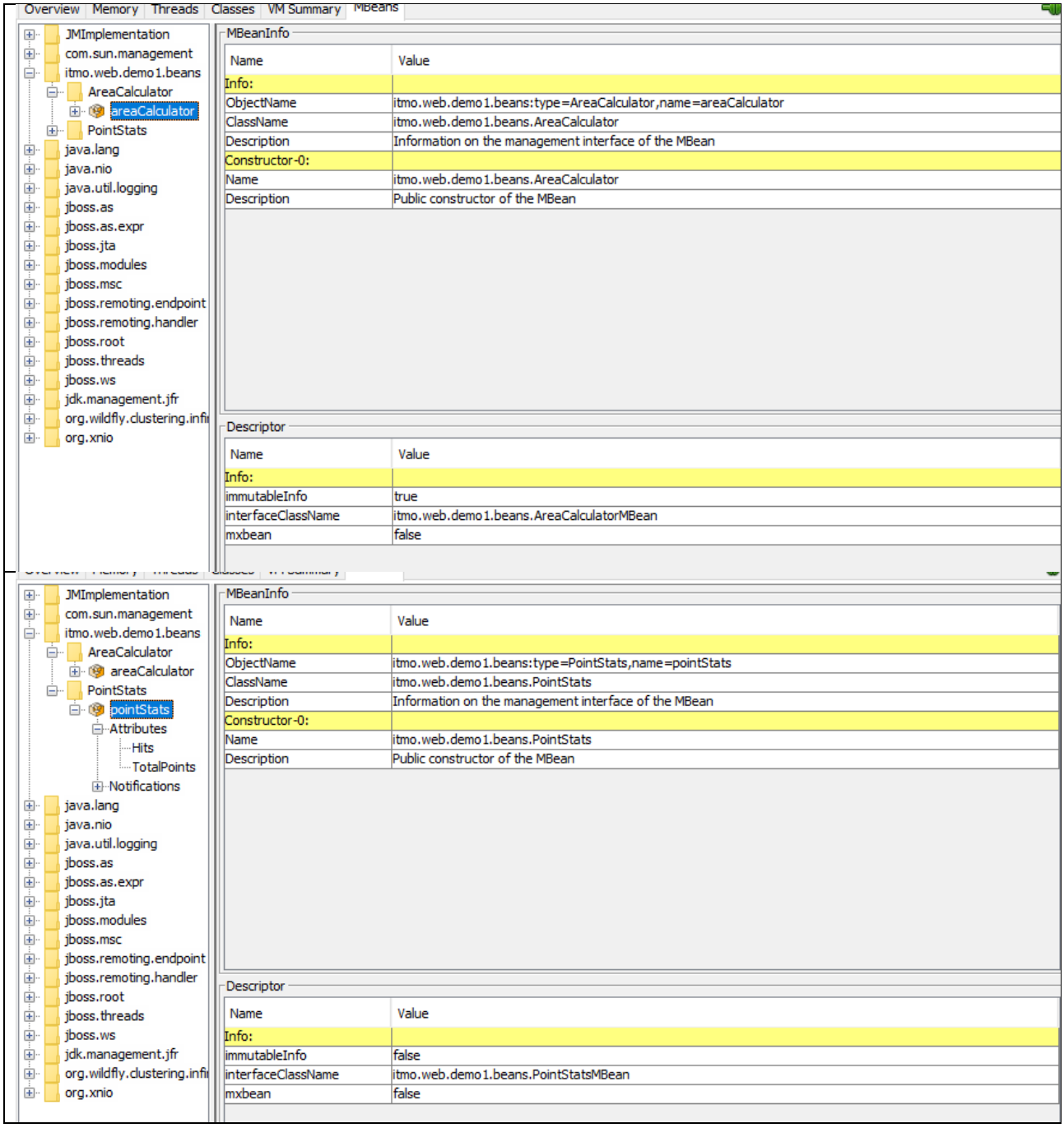


```java
package itmo.web.demo1.beans;

public interface PointStatsMBean {
    int getTotalPoints();
    int getHits();
}
```

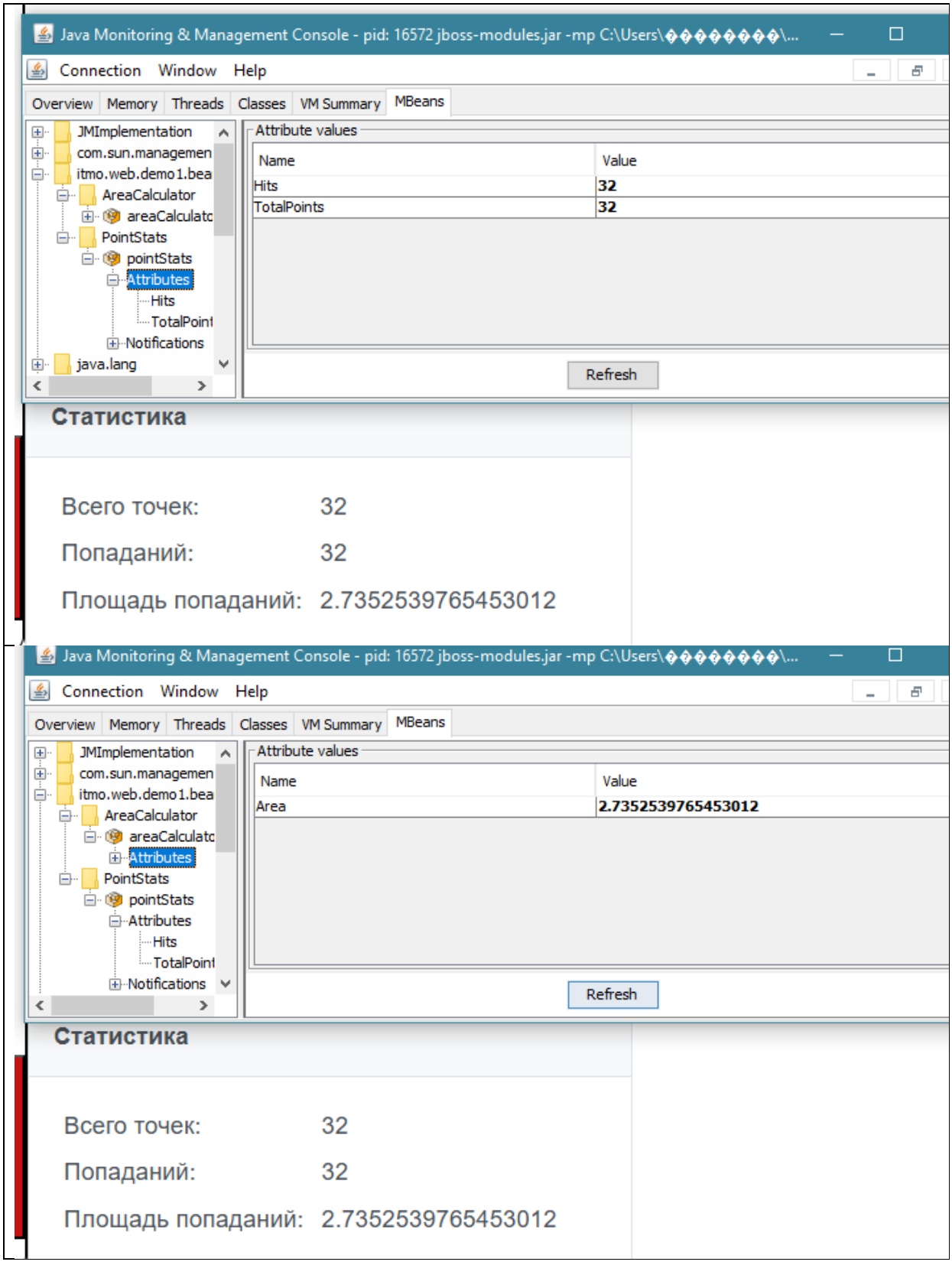
2. JConsole

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

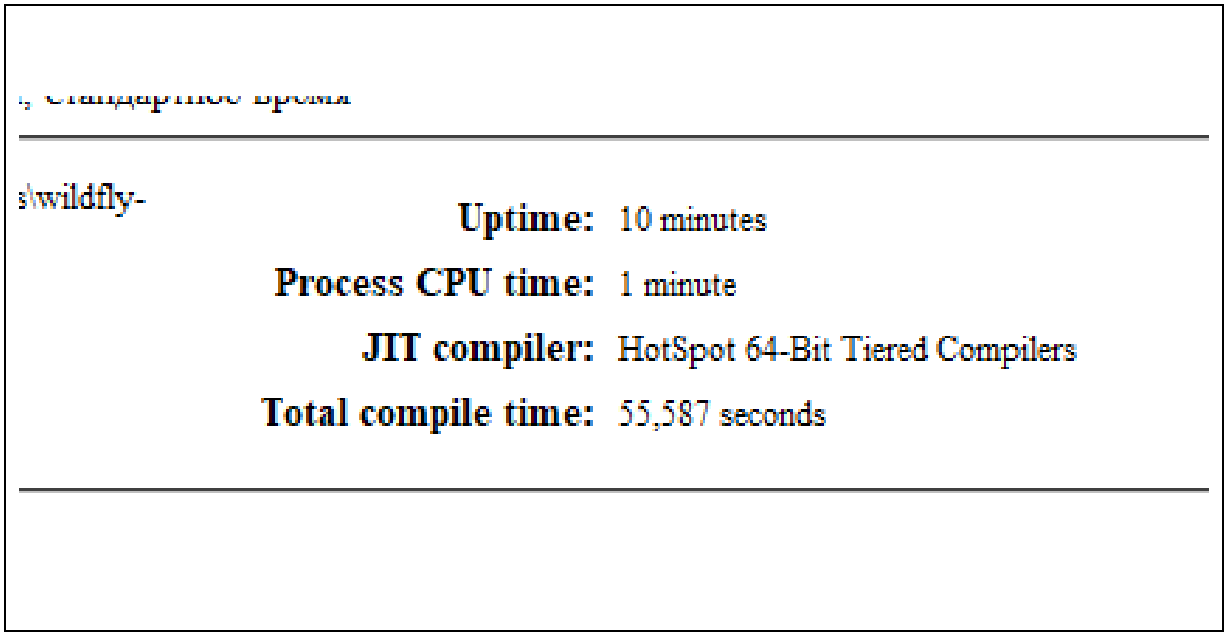
- **Метаданные** Mbean'ов:



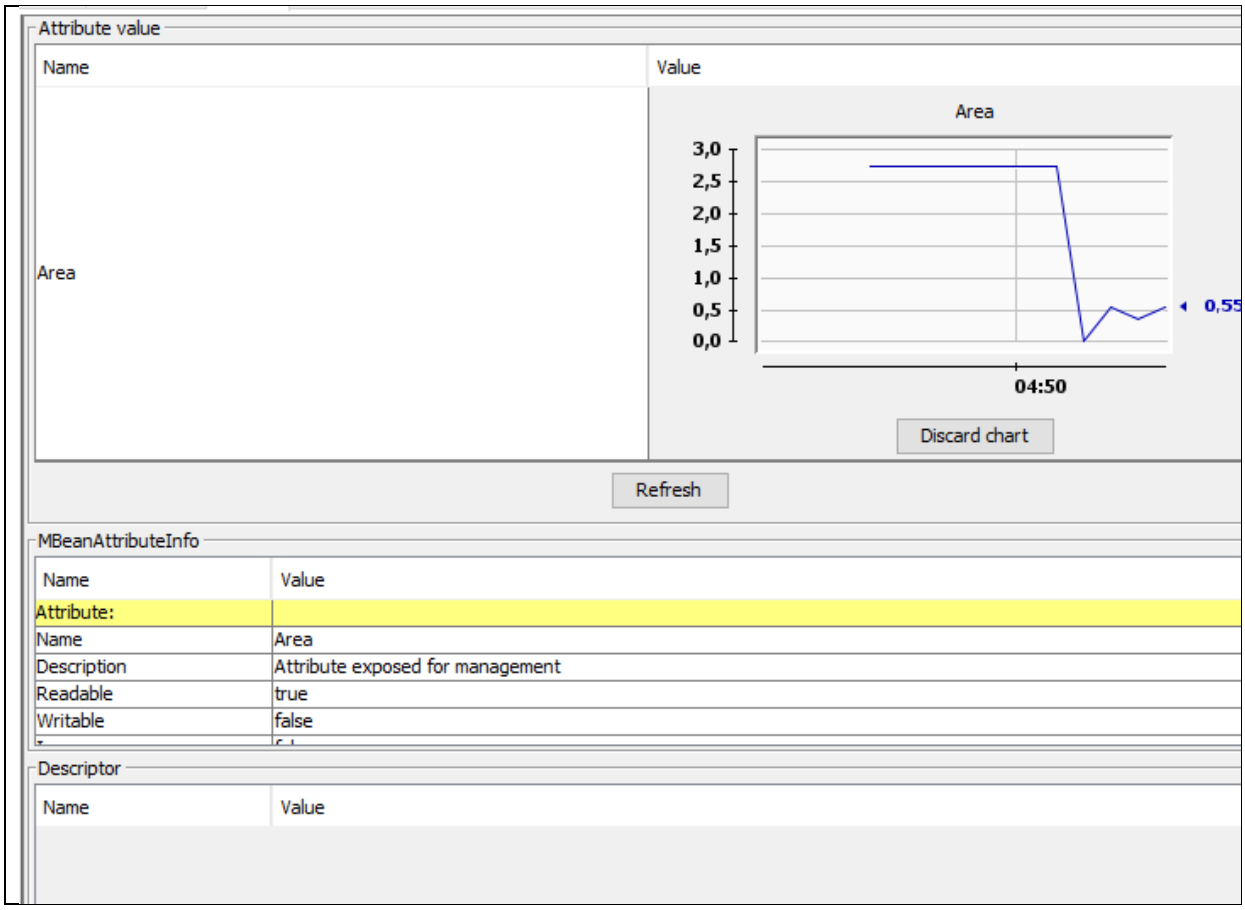
- **Показания** Mbean’ов:

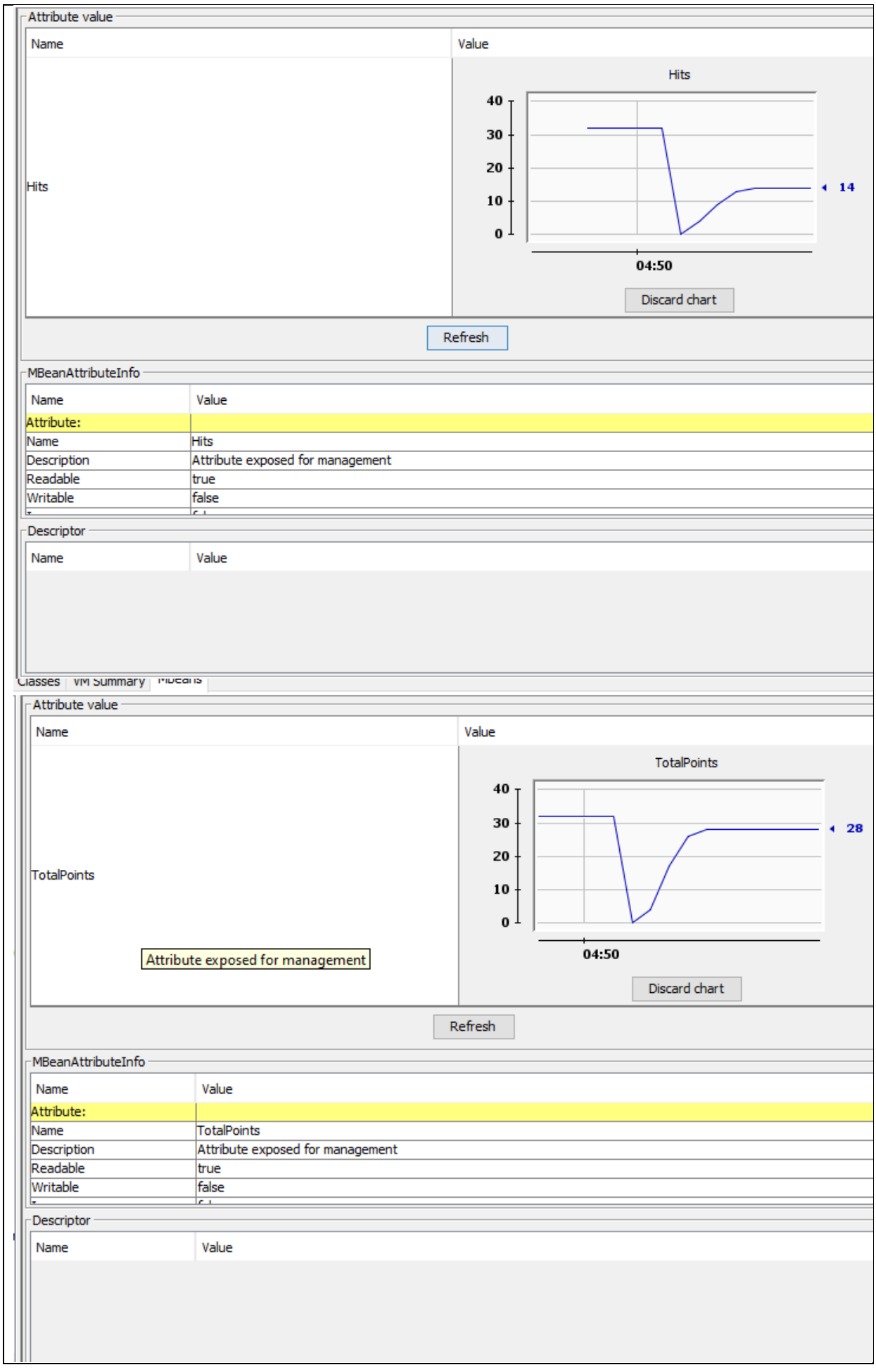


- **VM Summary** и время, прошедшее с момента запуска виртуальной машины:

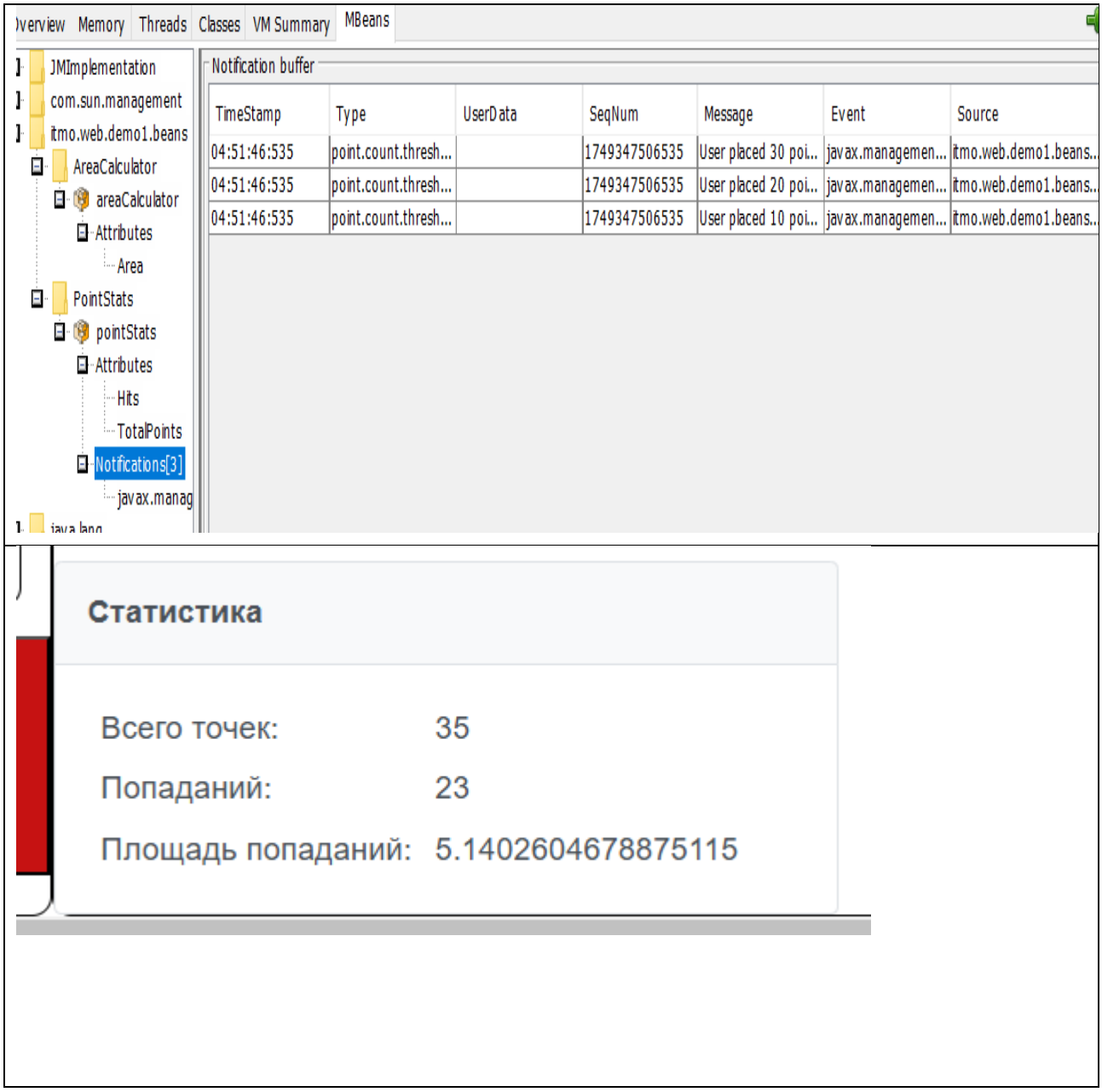


- **Графики**:




Attribute value
Name
Value
Hits
Hits
40
30
20
10
0
14
04:50
Discard chart
Refresh
MBeanAttributeInfo
Name
Value
Attribute:
Name
Hits
Description
Attribute exposed for management
Readable
true
Writable
false
Descriptor
Name
Value
Classes
VM Summary
MBeans
Attribute value
Name
Value
TotalPoints
TotalPoints
40
30
20
10
0
28
04:50
Attribute exposed for management
Discard chart
Refresh
MBeanAttributeInfo
Name
Value
Attribute:
Name
TotalPoints
Description
Attribute exposed for management
Readable
true
Writable
false
Descriptor
Name
Value

- **Уведомления**:



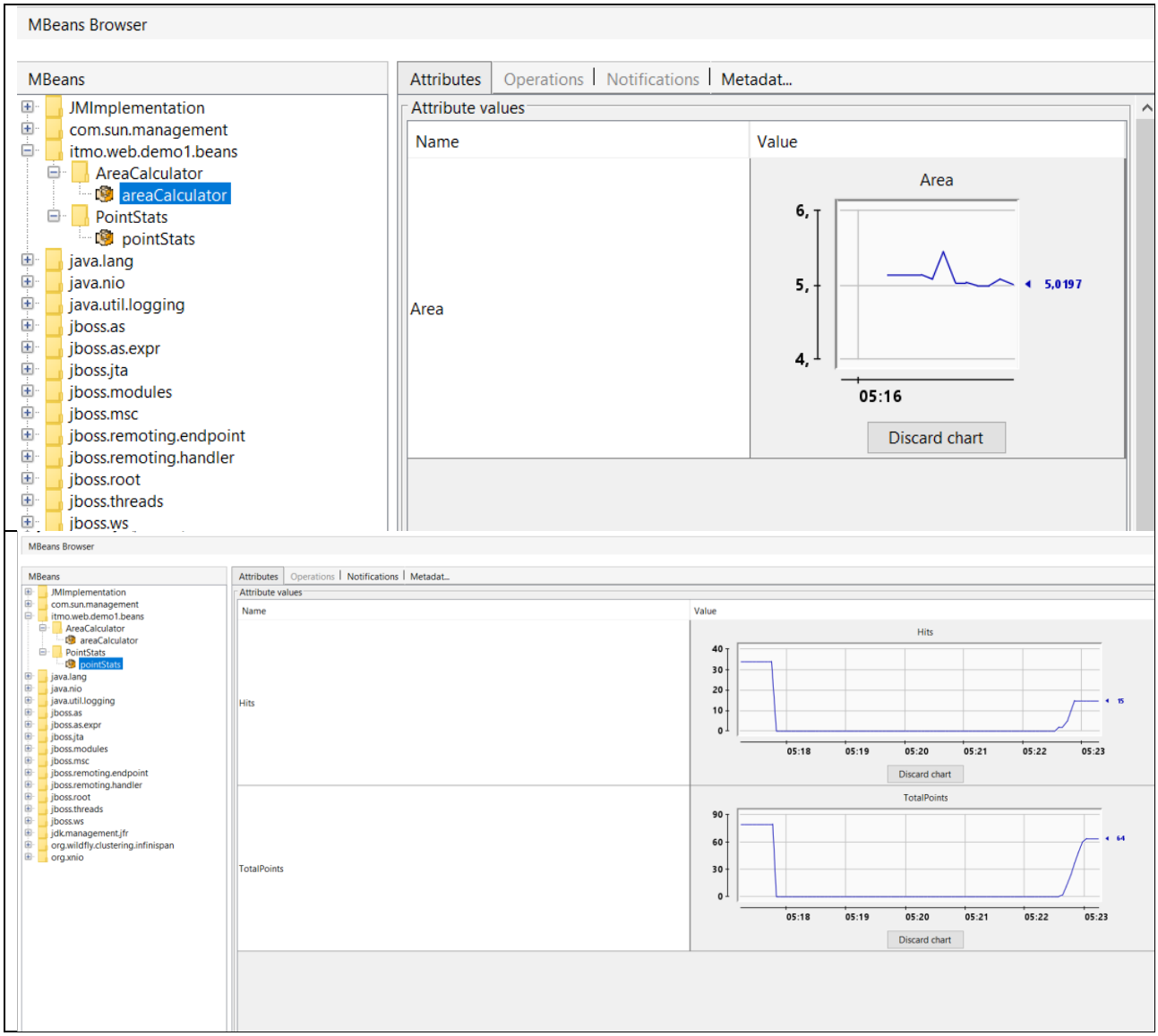
- **Выводы по результатам мониторинга**:

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

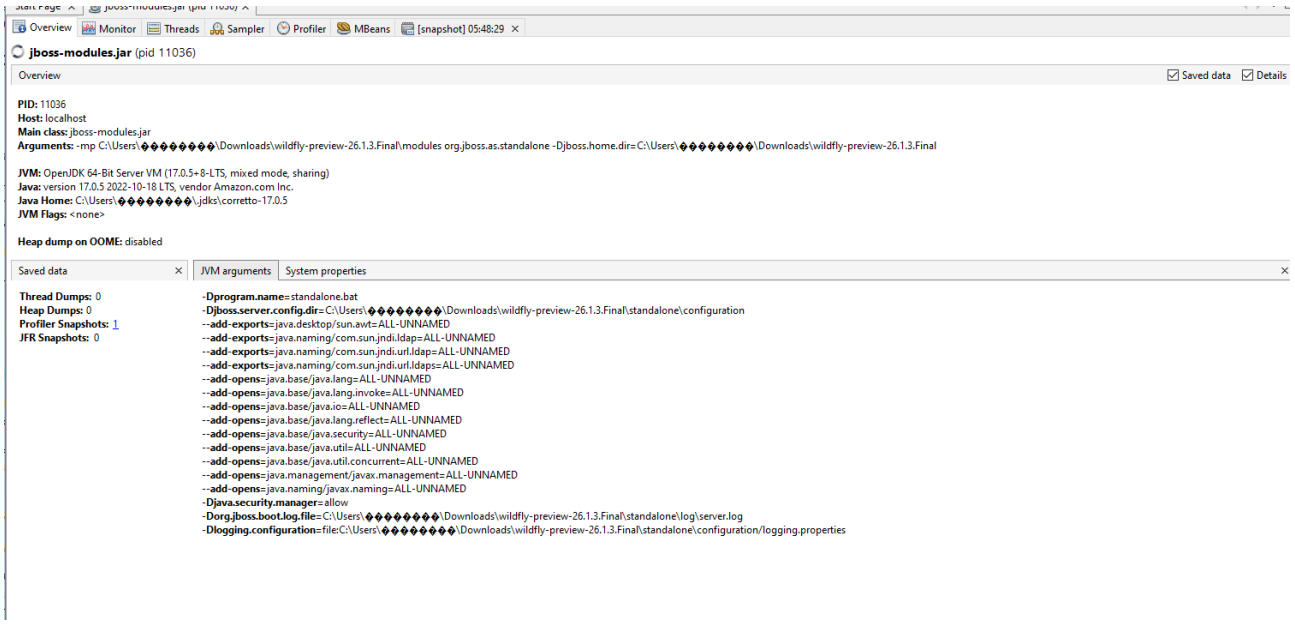
- Время работы JVM отображается через параметр Uptime на вкладке VM Summary.
- MBean PointStats был успешно зарегистрирован, атрибуты TotalPoints и Hits доступны во вкладке MBeans.
- При достижении количества точек, кратного 10, PointStats отправляет уведомления, которые успешно фиксируются JConsole.
- Таким образом, работоспособность системы мониторинга и оповещений подтверждена.

3. VisualVM

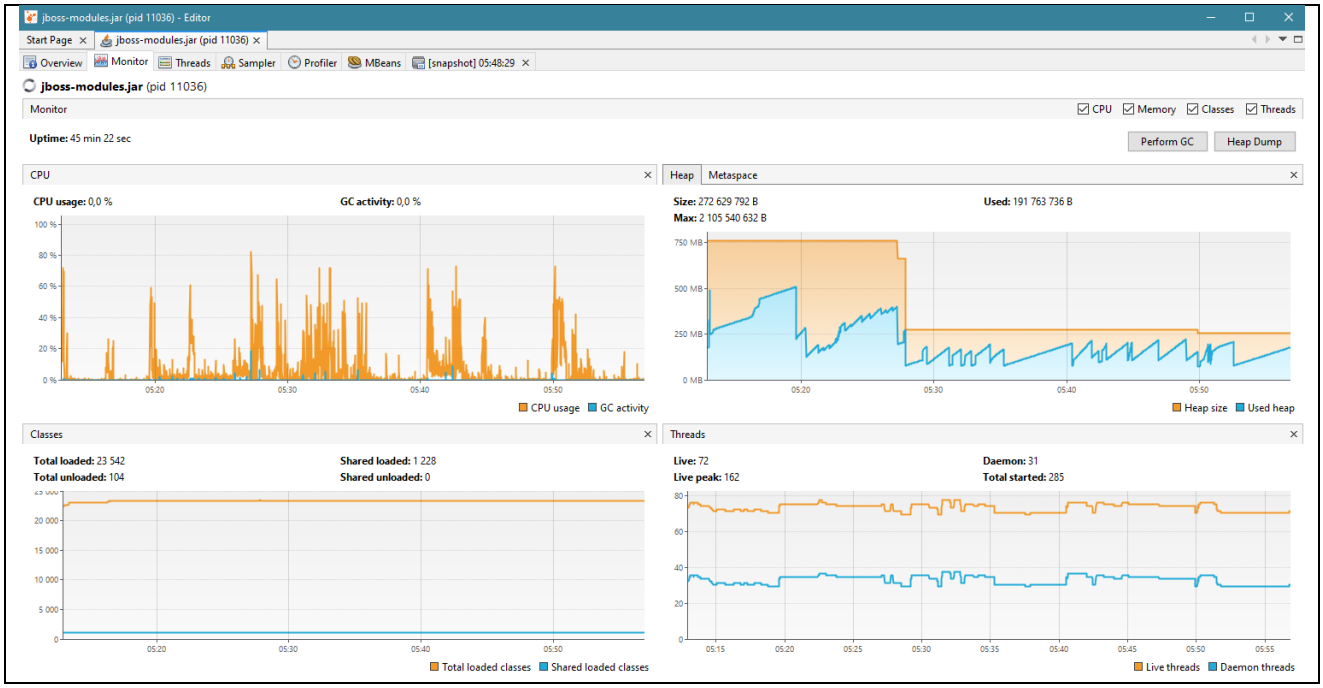
- **Снять график изменения показаний MBean-классов**:

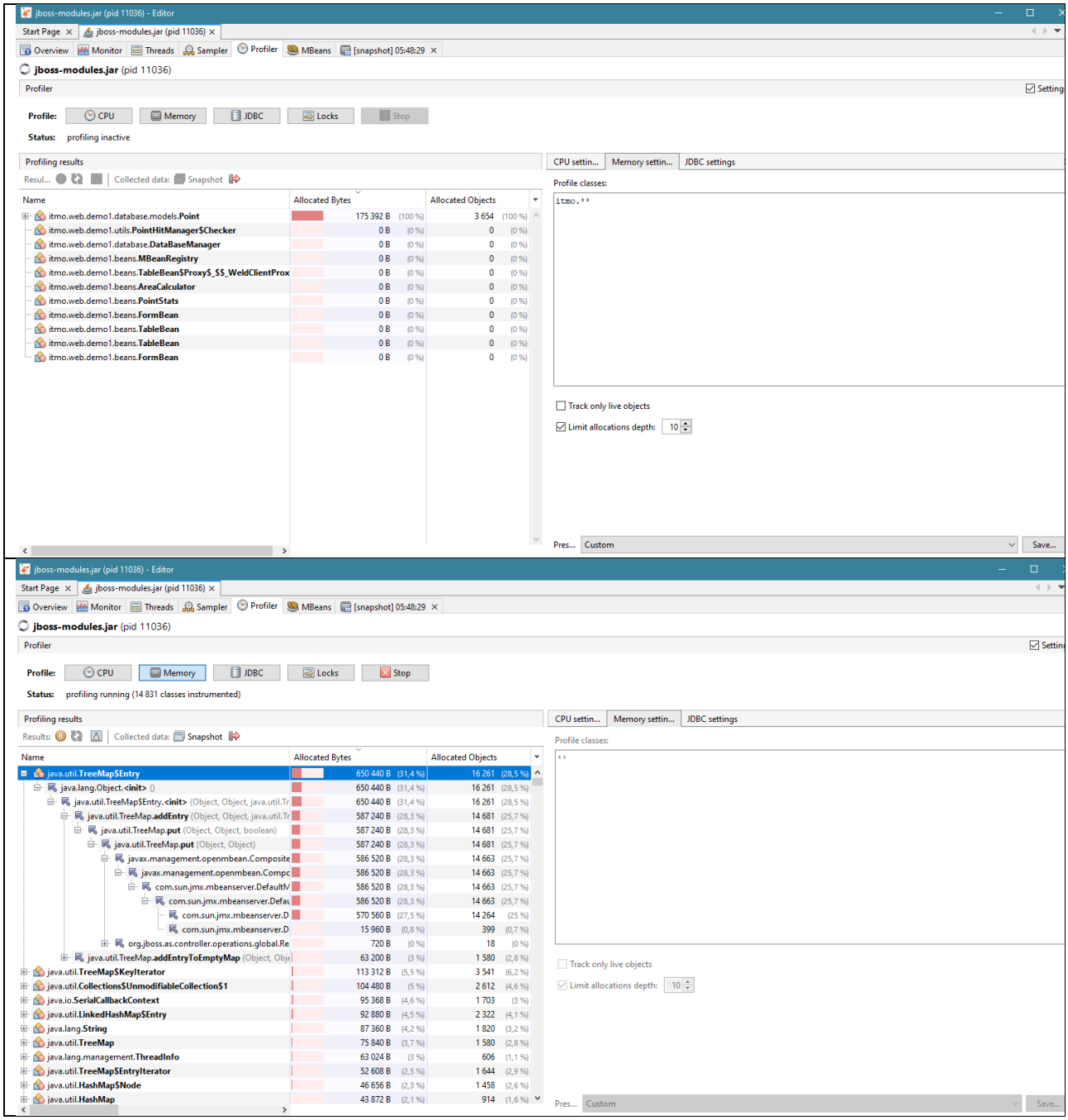


- **JVM Overview**:


Overview
Monitor
Threads
Sampler
Profiler
MBeans
[snapshot] 05:48:29
jboss-modules.jar (pid 11036)
Overview
Saved data
Details
PID: 11036
Host: localhost
Main class: jboss-modules.jar
Arguments: -mp C:\Users\●●●●●●●●\Downloads\wildfly-preview-26.1.3.Final\modules org.jboss.as.standalone -Djboss.home.dir=C:\Users\●●●●●●●●\Downloads\wildfly-preview-26.1.3.Final
JVM: OpenJDK 64-Bit Server VM (17.0.5+8-LTS, mixed mode, sharing)
Java: version 17.0.5 2022-10-18 LTS, vendor Amazon.com Inc.
Java Home: C:\Users\●●●●●●●●\.jdks\corretto-17.0.5
JVM Flags: <none>
Heap dump on OOME: disabled
Saved data
JVM arguments
System properties
Thread Dumps: 0
Heap Dumps: 0
Profiler Snapshots: 1
JFR Snapshots: 0
-Dprogram.name=standalone.bat
-Djboss.server.config.dir=C:\Users\●●●●●●●●\Downloads\wildfly-preview-26.1.3.Final\standalone\configuration
--add-exports=java.desktop/sun.awt=ALL-UNNAMED
--add-exports=java.naming/com.sun.jndi.ldap=ALL-UNNAMED
--add-exports=java.naming/com.sun.jndi.url.ldap=ALL-UNNAMED
--add-exports=java.naming/com.sun.jndi.url.ldaps=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
-Djava.security.manager=allow
-Dorg.jboss.boot.log.file=C:\Users\●●●●●●●●\Downloads\wildfly-preview-26.1.3.Final\standalone\log\server.log
-Dlogging.configuration=file:C:\Users\●●●●●●●●\Downloads\wildfly-preview-26.1.3.Final\standalone\configuration/logging.properties

- **Мониторинг**:


jboss-modules.jar (pid 11036) - Editor
Start Page
jboss-modules.jar (pid 11036)
Overview
Monitor
Threads
Sampler
Profiler
MBeans
[snapshot] 05:48:29
jboss-modules.jar (pid 11036)
Monitor
CPU
Memory
Classes
Threads
Uptime: 45 min 22 sec
Perform GC
Heap Dump
CPU
CPU usage: 0,0 %
GC activity: 0,0 %
100 %
80 %
60 %
40 %
20 %
0 %
05:20
05:30
05:40
05:50
CPU usage
GC activity
Heap
Metaspace
Size: 272 629 792 B
Used: 191 763 736 B
Max: 2 105 540 632 B
750 MB
500 MB
250 MB
0 MB
05:20
05:30
05:40
05:50
Heap size
Used heap
Classes
Total loaded: 23 542
Shared loaded: 1 228
Total unloaded: 104
Shared unloaded: 0
20 000
15 000
10 000
5 000
0
05:20
05:30
05:40
05:50
Total loaded classes
Shared loaded classes
Threads
Live: 72
Daemon: 31
Live peak: 162
Total started: 285
80
60
40
20
0
05:15
05:20
05:25
05:30
05:35
05:40
05:45
05:50
05:55
Live threads
Daemon threads



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

Изменения показаний MBean-классов с течением времени:

● Анализ изменения показаний MBean-классов с течением времени:

В ходе мониторинга приложения с использованием вкладки MBeans в VisualVM были получены графики времени от двух MBean-классов: AreaCalculator и PointStats.

Атрибут Area демонстрирует динамику изменения площади попаданий в фигуру, формируемую пользователем. На графике отчётливо видно колебание значения площади в пределах от примерно 5.0 до 5.5, что указывает на то, что пользователь активно добавлял новые точки, тем самым влияя на общую форму и площадь фигуры.

Атрибуты Hits и TotalPoints позволяют отслеживать общее число добавленных пользователем точек и число точек, попавших в область. На графиках видно два характерных изменения: резкое снижение значений Hits и TotalPoints в 05:18 связано, вероятно, с полной очисткой базы данных; в 05:22 происходит рост значений, что указывает на повторное добавление точек пользователем.

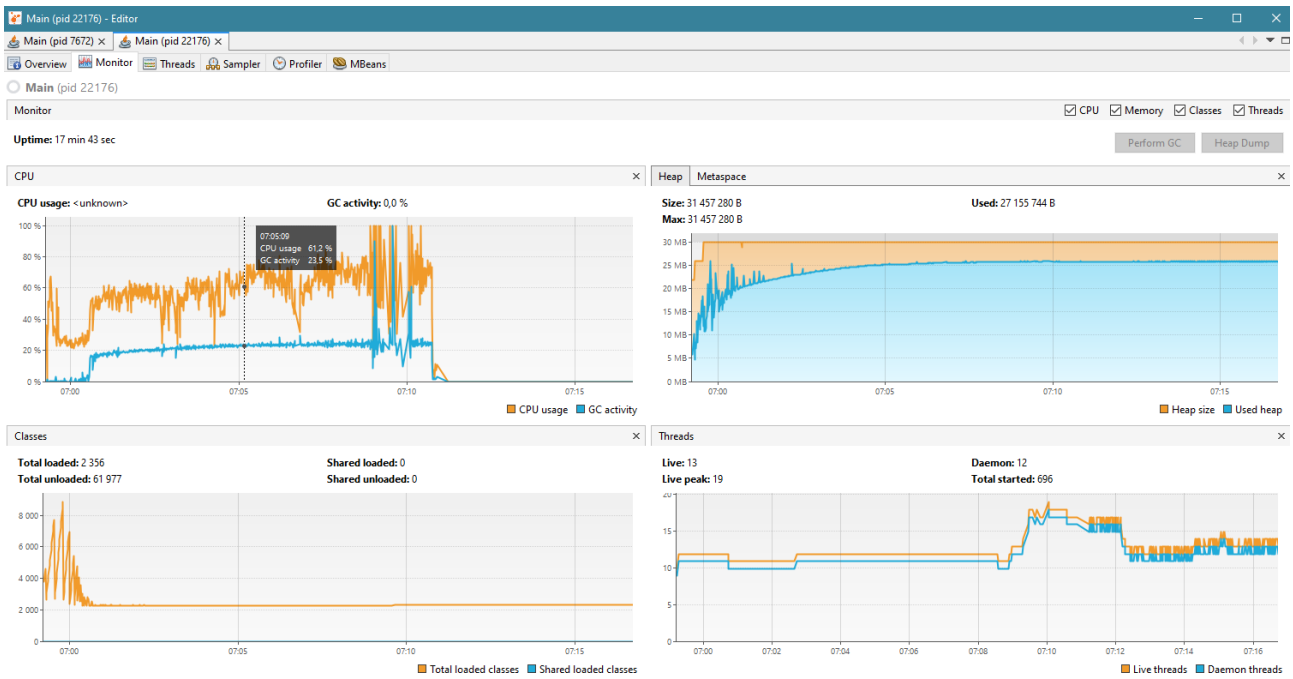
По результатам анализа памяти, проведённого через вкладку Profiler → Memory, установлено, что наибольшее количество памяти в пределах всей JVM занимают стандартные структуры: класс java.util.TreeMap$Entry занимает 650 КБ, что составляет 81,4 % всех зафиксированных аллокаций. Эти объекты относятся к внутренним структурам данных платформы JBoss/WildFly и не являются пользовательскими.

Среди пользовательских классов наибольшую нагрузку на память создаёт класс itmo.web.demo1.database.models.Point. Всего было создано 3 654 объекта, общий объём занимаемой ими памяти составляет 175 392 байта. Это логично, так как данный класс представляет координаты точек, создаваемых пользователем через интерфейс. Объекты Point хранятся в экземпляре класса TableBean, где используется коллекция List<Point>. Это делает TableBean косвенно ответственным за потребление основной части памяти пользовательского уровня.

Поведение MBean-метрик подтверждает динамику взаимодействия пользователя с приложением, включая удаление и повторное добавление данных. Основным источником потребления памяти на уровне пользовательского кода является класс Point, логично отражающий назначение приложения — работу с координатными точками. Общая нагрузка от пользовательских классов по сравнению с внутренними системными структурами минимальна и не создаёт проблем с производительностью.

4. Исследование программы на утечки памяти

Я запустил программу на Java version 1.8.0_452 и ограничил размер кучи -Xmx30m. Дальше смотрим на графики.



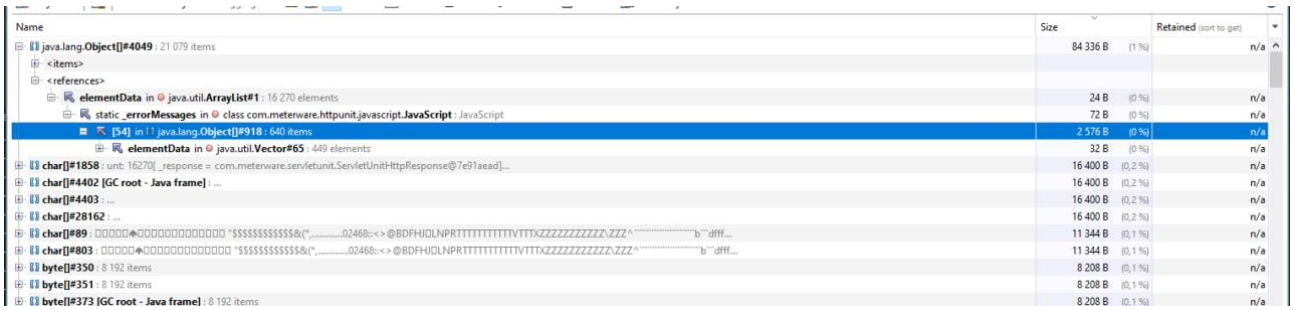
Можем увидеть что в примерно в 7:10 программа легла на 10 минуте. Активность GB 0%. Смотрим на график кучи, видим, что размер используемой кучи постепенно растет, в то время как GB не работает. Явно видим утечку кучи. Смотрим дальше в терминал.

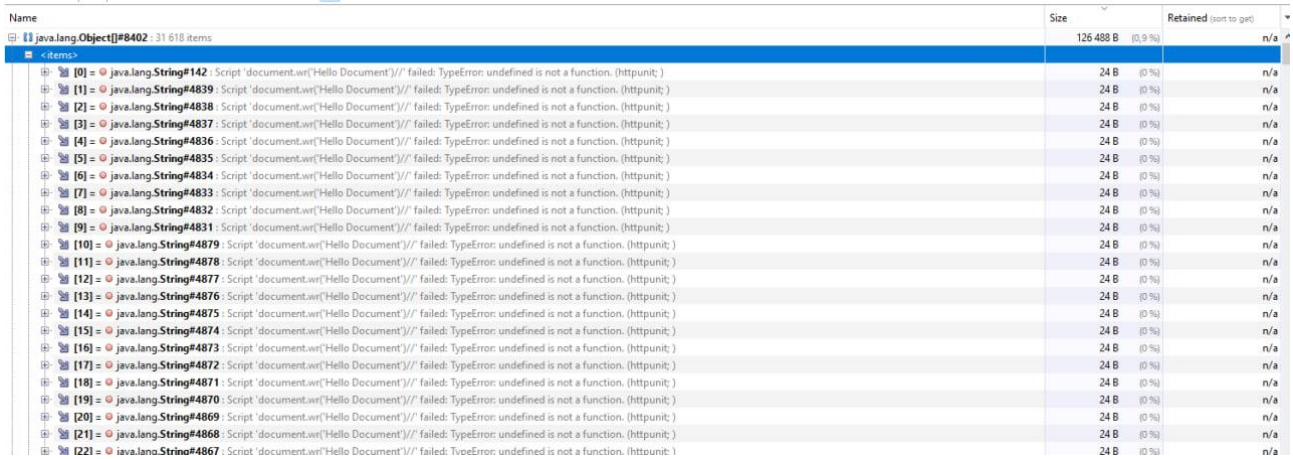
```
Count: 61773[ _response = com.meterware.servletunit.ServletUnitHttpResponse@5ff24c3c]
Count: 61774[ _response = com.meterware.servletunit.ServletUnitHttpResponse@1d0c0ee8]
Profiler Agent: Waiting for connection on port 5140 (Protocol version: 21)
Count: 61775[ _response = com.meterware.servletunit.ServletUnitHttpResponse@f1807c2]
Profiler Agent: Established connection with the tool
Exception in thread "*** Profiler Agent Special Execution Thread 1" java.lang.OutOfMemoryError Create breakpoint : Java heap space
    at java.util.zip.DeflaterOutputStream.<init>(DeflaterOutputStream.java:109)
    at java.util.zip.GZIPOutputStream.<init>(GZIPOutputStream.java:90)
    at java.util.zip.GZIPOutputStream.<init>(GZIPOutputStream.java:67)
    at org.graalvm.visualvm.lib.jfluid.wireprotocol.RootClassLoadedCommand.writeObject(RootClassLoadedCommand.java:187)
    at org.graalvm.visualvm.lib.jfluid.wireprotocol.WireIO.sendComplexCommand(WireIO.java:295)
    at org.graalvm.visualvm.lib.jfluid.server.ProfilerServer.sendComplexCmdToClient(ProfilerServer.java:696)
    at org.graalvm.visualvm.lib.jfluid.server.ProfilerInterface$HFIRIThread.run(ProfilerInterface.java:74)
Count: 61776[ _response = com.meterware.servletunit.ServletUnitHttpResponse@68df1175]
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
Exception in thread "*** JFluid Monitor thread ***" java.lang.OutOfMemoryError: Java heap space
Exception in thread "RMI TCP Connection(idle)" java.lang.OutOfMemoryError: Java heap space
Exception in thread "RMI TCP Connection(idle)" java.lang.OutOfMemoryError: Java heap space
Exception in thread "RMI TCP Connection(idle)" java.lang.OutOfMemoryError: Java heap space
Exception in thread "RMI TCP Connection(idle)" java.lang.OutOfMemoryError: Java heap space
```

Программа упала почти на 61 775-й итерации в связи с ошибкой:

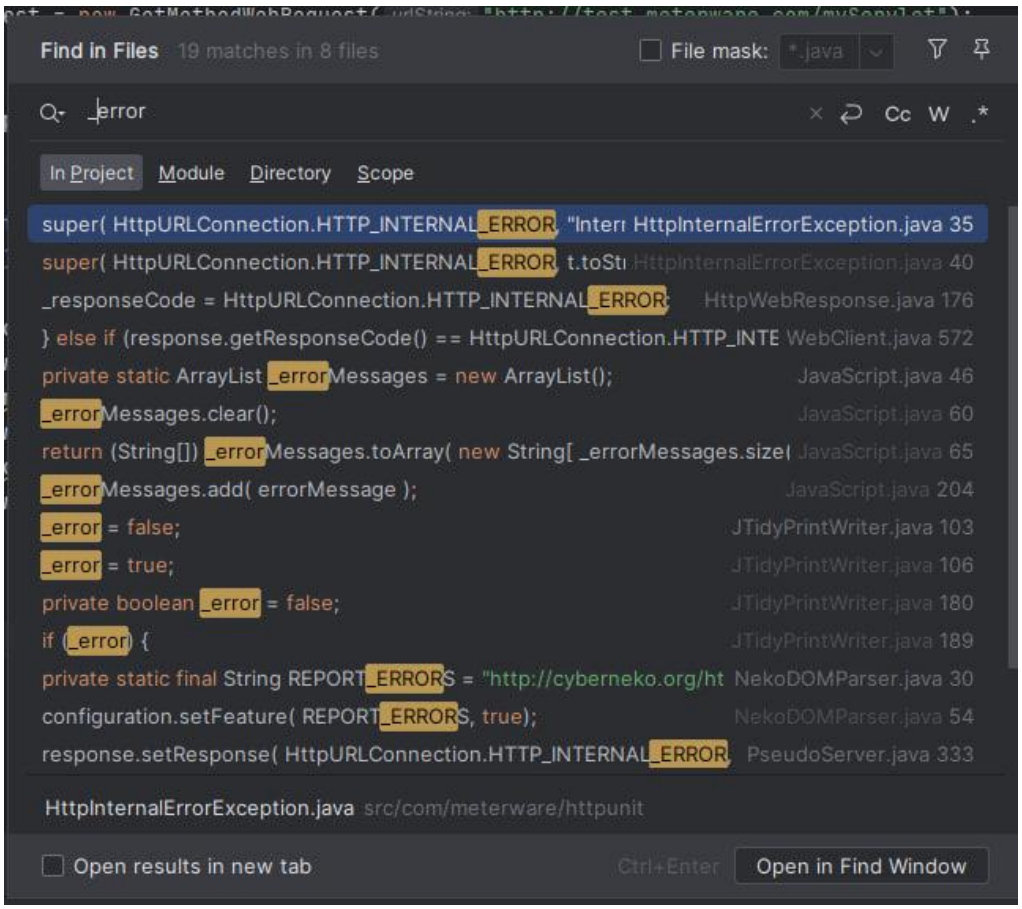
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space.

Запустим снова и посмотрим, что у нас занимает память





Изучение heap dump выявляет наличие крупного массива объектов Object[], включающего более 30 000 строк с повторяющимися сообщениями об ошибке JavaScript: Script 'document.wr(...)' failed: TypeError: undefined is not a function. Эти строки являются результатом попытки клиента обработать HTML, возвращаемый от сервера, в котором содержится преднамеренно некорректный JavaScript (document.wr(...) вместо document.write(...)). Такой код размещён в теле метода doGet() сервлета, и с каждым запросом он вызывает исключение на стороне HttpUnit, порождая всё новые строки ошибок.




```
194
195         private void handleScriptException( Exception e, String badScript ) {
196             final String errorMessage = badScript + " failed: " + e;
197             if (!(e instanceof EcmaError) && !(e instanceof EvaluatorException)) {
198                 e.printStackTrace();
199                 throw new RuntimeException( errorMessage );
200             } else if (isThrowExceptionsOnError()) {
201                 e.printStackTrace();
202                 throw new ScriptException( errorMessage );
203             } else {
204                 _errorMessages.add( errorMessage );
205             }
206         }
207
```

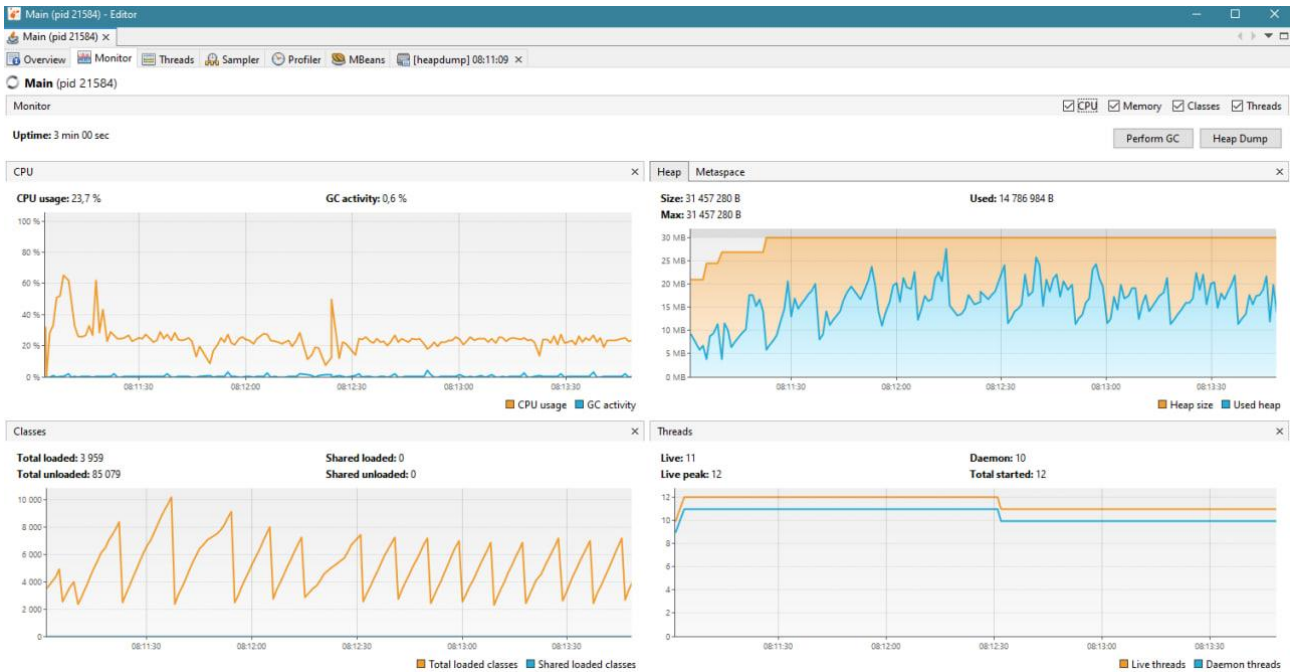
```java
/**
 * Clears the accumulated script error messages.
 */
public static void clearScriptErrorMessages() {
    getScriptingEngine().clearErrorMessages();
}
```

```java
 *
 * @author gs145266
 */
public class HelloWorld extends HttpServlet {

    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        response.setContentType("text/html");
        out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
                                    "Transitional//EN\">\n" +
                "<HTML>\n" +
                "<HEAD><TITLE>Hello World</TITLE></HEAD>\n" +
                "<BODY>\n" +
                "<H1>Hello World</H1>\n");
        out.println("<script language=\"JavaScript\"type=\"text/javascript\">");
        out.println("<!--");
        out.println("document.wr('Hello Document')");
        out.println("//-->");
        out.println("</script>");
        out.println("</BODY></HTML>");
    }
}
```

```
while (true) {
    WebResponse response = sc.getResponse(request);
    System.out.println("Count: " + number++ + response);
    HttpUnitOptions.clearScriptErrorMessages();
```

Прослеживая путь объектов в heap, становится ясно, что все эти строки накапливаются внутри статического поля _errorMessages класса JavaScript, доступного через цепочку ArrayList → Vector → ServletUnitHttpResponse → String. Поскольку поле является статическим, оно не подлежит сборке мусора, и все ошибки, записанные через errorMessages.add(...), остаются в памяти на протяжении всей жизни программы.

Поиск по коду проекта подтверждает, что ошибки действительно сохраняются в массив через метод handleScriptException, вызываемый при обработке скриптов. Однако библиотека предоставляет специальный метод HttpUnitOptions.clearScriptErrorMessages(), который очищает накопленные сообщения об ошибках. Добавив вызов этого метода в конце каждой итерации цикла, где отправляется HTTP-запрос, удаётся устранить рост памяти и стабилизировать поведение программы. Это решение подтверждается повторным запуском и наблюдением в VisualVM, где после включения очистки рост heap прекращается, а программа успешно продолжает выполнение без утечек.



Программа работает стабильно и не падает с `OutOfMemoryException`.

2. Вывод

В ходе выполнения лабораторной работы я освоил применение MBean-компонентов в составе веб-приложений, научился использовать инструменты JConsole и VisualVM для анализа состояния приложения во время выполнения, а также приобрёл практические навыки выявления и устранения утечек памяти на основе собранных диагностических данных.