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: 45585

**Преподаватель:**   
Осипов Святослав Владимирович

**Выполнил:**

Фан Нгок Туан

**Группа:** Р3221

Санкт-Петербург, 2025

|  |
| --- |
| **Table of Contents** |

Contents

[1. Задание 2](#_Toc196525867)

[2. MBean 3](#_Toc196525868)

[1. PointStatsMBean 3](#_Toc196525869)

[2. PointStats 3](#_Toc196525870)

[3. ShapeAreaMBean 4](#_Toc196525871)

[4. ShapeArea 4](#_Toc196525872)

[3. JConsole 5](#_Toc196525873)

[4. VisualVM 6](#_Toc196525874)

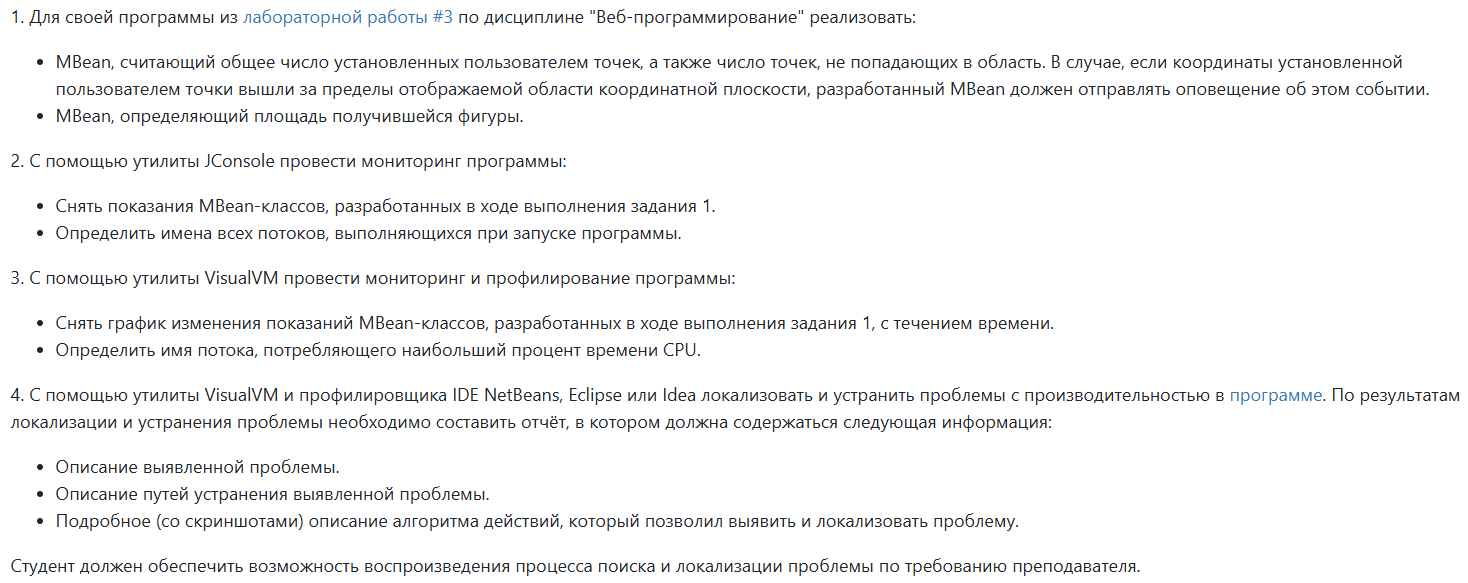
[1. Показания VisualVM 6](#_Toc196525875)

[2. Определить имя потока, потребляющего наибольший процент времени CPU. 7](#_Toc196525876)

[5. Поиск утечки памяти 7](#_Toc196525877)

[6. Вывод 13](#_Toc196525878)

## **1. Задание**



## 2. MBean

### 1. PointStatsMBean

public interface PointStatsMBean {  
 int getTotalPoints();  
 int getTotalMisses();  
 void checkForOutOfBounds(double x, double y, double r);  
}

### 2. PointStats

@Component  
public class PointStats extends NotificationBroadcasterSupport implements PointStatsMBean, Serializable {  
 private final AtomicInteger totalPoints = new AtomicInteger();  
 private final AtomicInteger totalMisses = new AtomicInteger();  
 private long sequenceNumber = 1;  
  
 @Override  
 public int getTotalPoints() {  
 return totalPoints.get();  
 }  
  
 @Override  
 public int getTotalMisses() {  
 return totalMisses.get();  
 }  
  
 @Override  
 public void checkForOutOfBounds(double x, double y, double r) {  
 if (x < -r - 1 || x > r + 1 || y < -r - 1 || y > r + 1) {  
 Notification notification = new Notification(  
 "point.outOfCoordinatesPlane",  
 this,  
 sequenceNumber++,  
 System.*currentTimeMillis*(),  
 "The point (" + x + ", " + y + ") is out of coordinate plane for r=" + r  
 );  
 sendNotification(notification);  
 }  
 }  
  
 public void updateStats(boolean hit, double x, double y, double r) {  
 totalPoints.incrementAndGet();  
 if (!hit) {  
 totalMisses.incrementAndGet();  
 }  
 checkForOutOfBounds(x, y, r);  
 }  
  
 @Override  
 public MBeanNotificationInfo[] getNotificationInfo() {  
 String[] types = new String[]{"point.outOfCoordinatesPlane"};  
 String name = Notification.class.getName();  
 String description = "Point is out of the displayed area";  
 return new MBeanNotificationInfo[]{  
 new MBeanNotificationInfo(types, name, description)  
 };  
 }  
}

### 3. ShapeAreaMBean

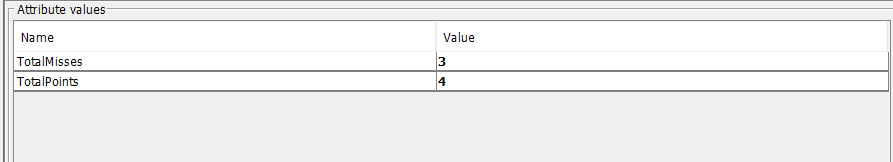
public interface ShapeAreaMBean {  
 void updateShapeArea(double r);  
 double getArea();  
}

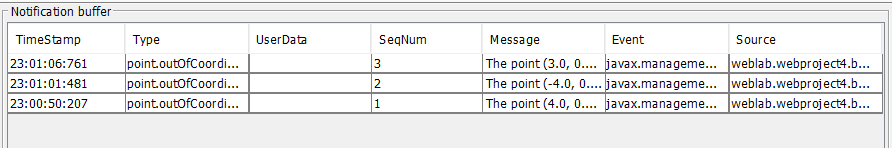
### 4. ShapeArea

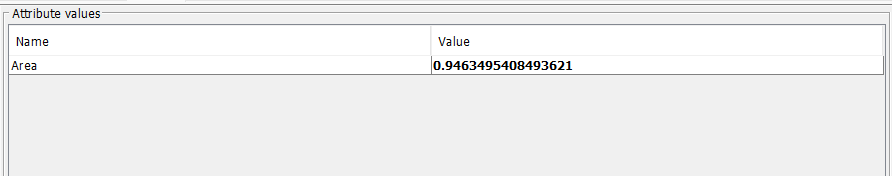
@Component  
public class ShapeArea implements ShapeAreaMBean, Serializable {  
 private double area;  
 @Override  
 public void updateShapeArea(double r) {  
 area = calcTriangleArea(r) + calcRectangleArea(r) + calcCircleArea(r);  
 }  
  
 @Override  
 public double getArea() {  
 return area;  
 }  
  
 private double calcTriangleArea(double r) {  
 return 0.5 \* (r / 2.0) \* r;  
 }  
  
 private double calcRectangleArea(double r) {  
 return r \* (r / 2.0);  
 }  
  
 private double calcCircleArea(double r) {  
 return 0.25 \* Math.*PI* \* Math.*pow*(r / 2.0, 2);  
 }  
}

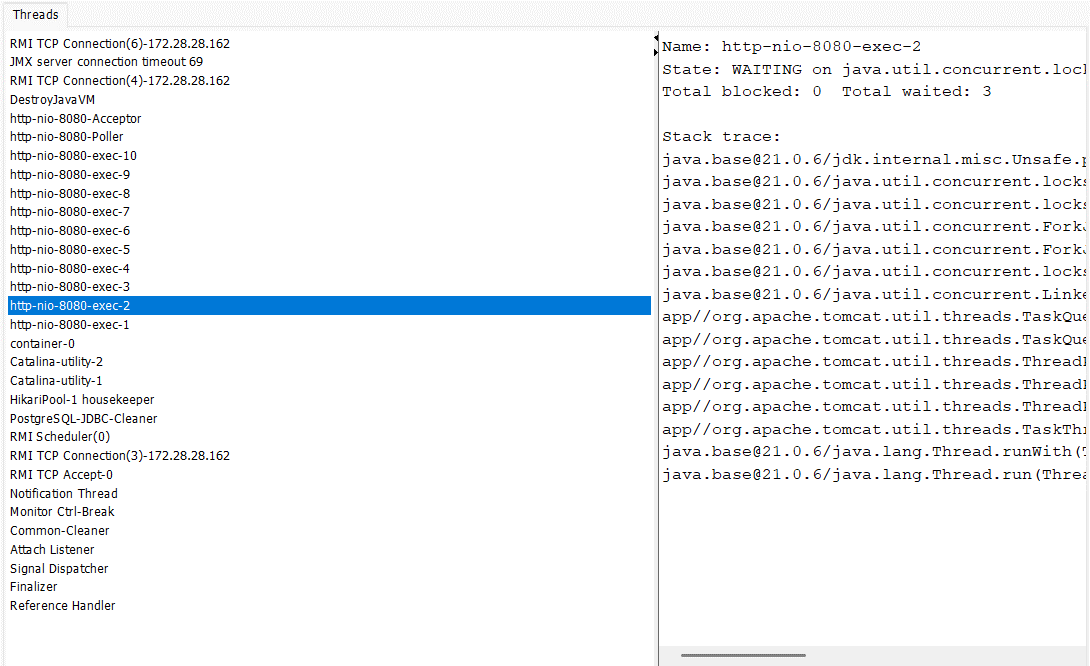
## 3. JConsole

1. Показания JConsole :



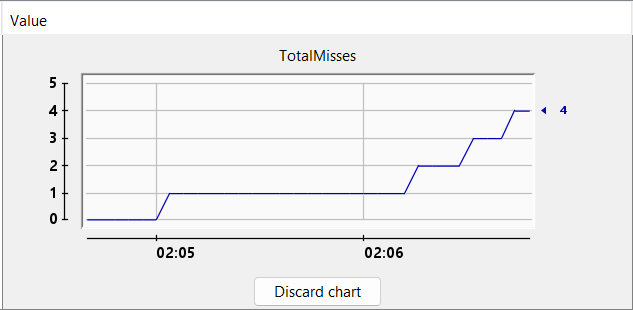


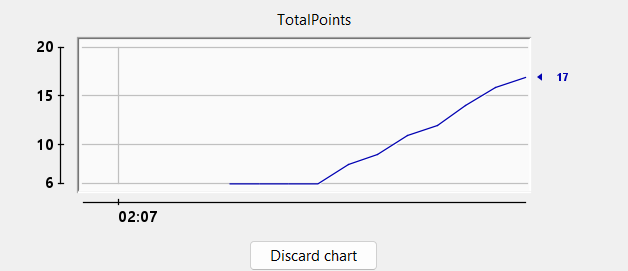


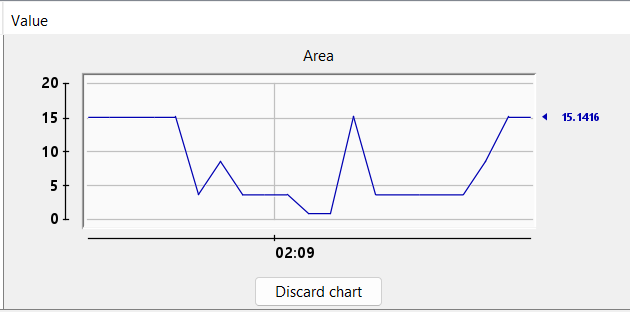
2. Определить имена всех потоков, выполняющихся при запуске программы. 

## 4. VisualVM

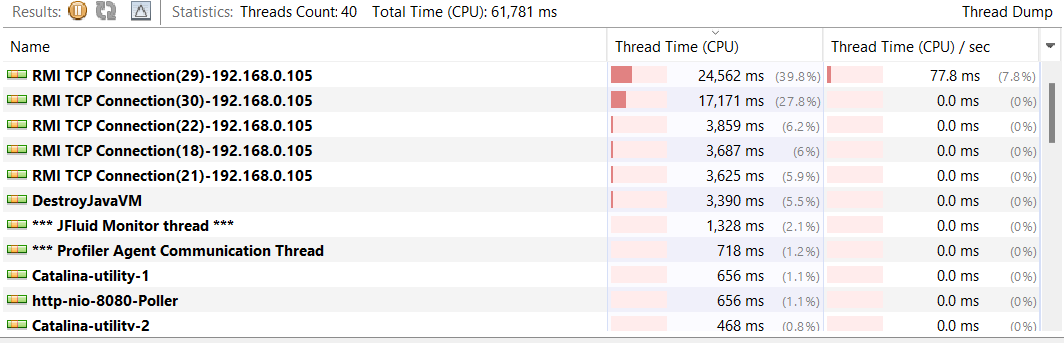
### 1. Показания VisualVM







### 2. Определить имя потока, потребляющего наибольший процент времени CPU.



## 5. Поиск утечки памяти

1. Подготовка окружения :

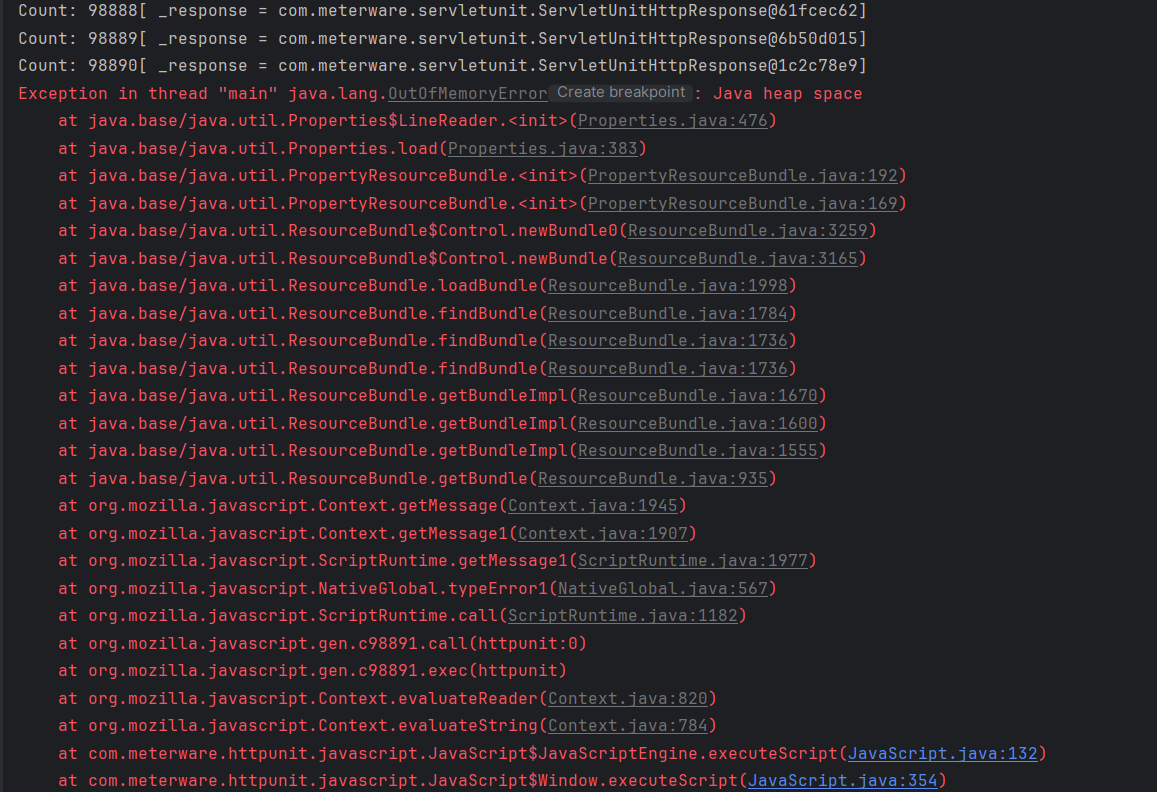
Основная часть : бесконечный цикл с запросами

while (true) {  
 WebResponse response = sc.getResponse(request);  
 System.*out*.println("Count: " + number++ + response);  
 java.lang.Thread.*sleep*(200);  
}

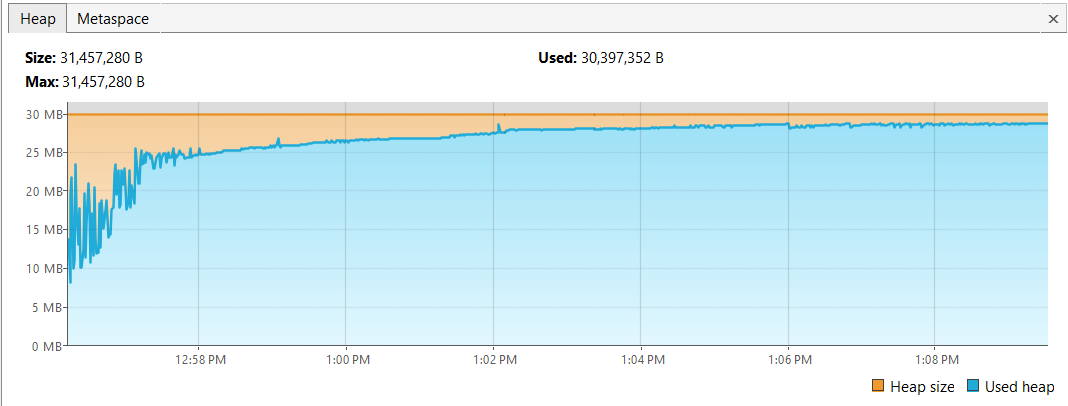
Чтобы ускорить выбрасывание OutOfMemoryError:

* Установим задержку между запросами (Thread.sleep) в 0 мс
* Установим максимальный размер кучи на 30 Мб

1. Анализ используемой памяти

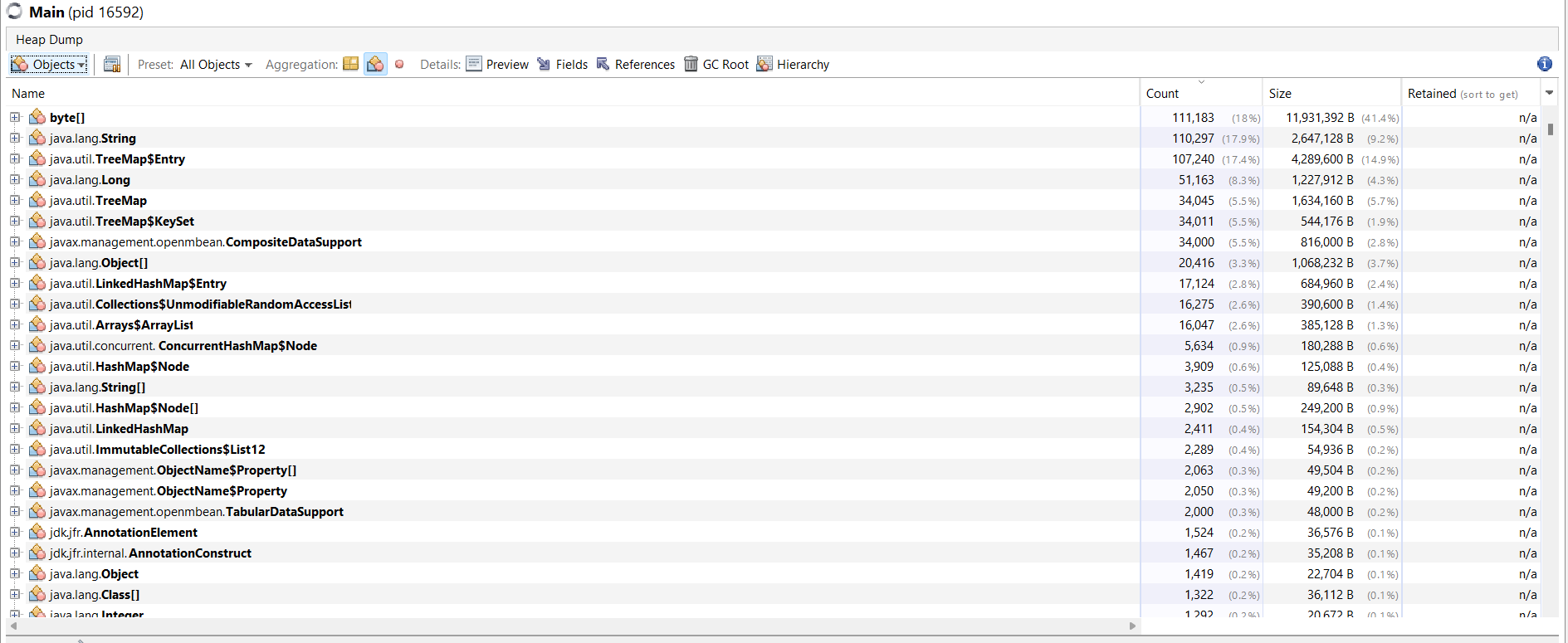
Через несколько минут после запуска программы выбрасывается исключение:  
  


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

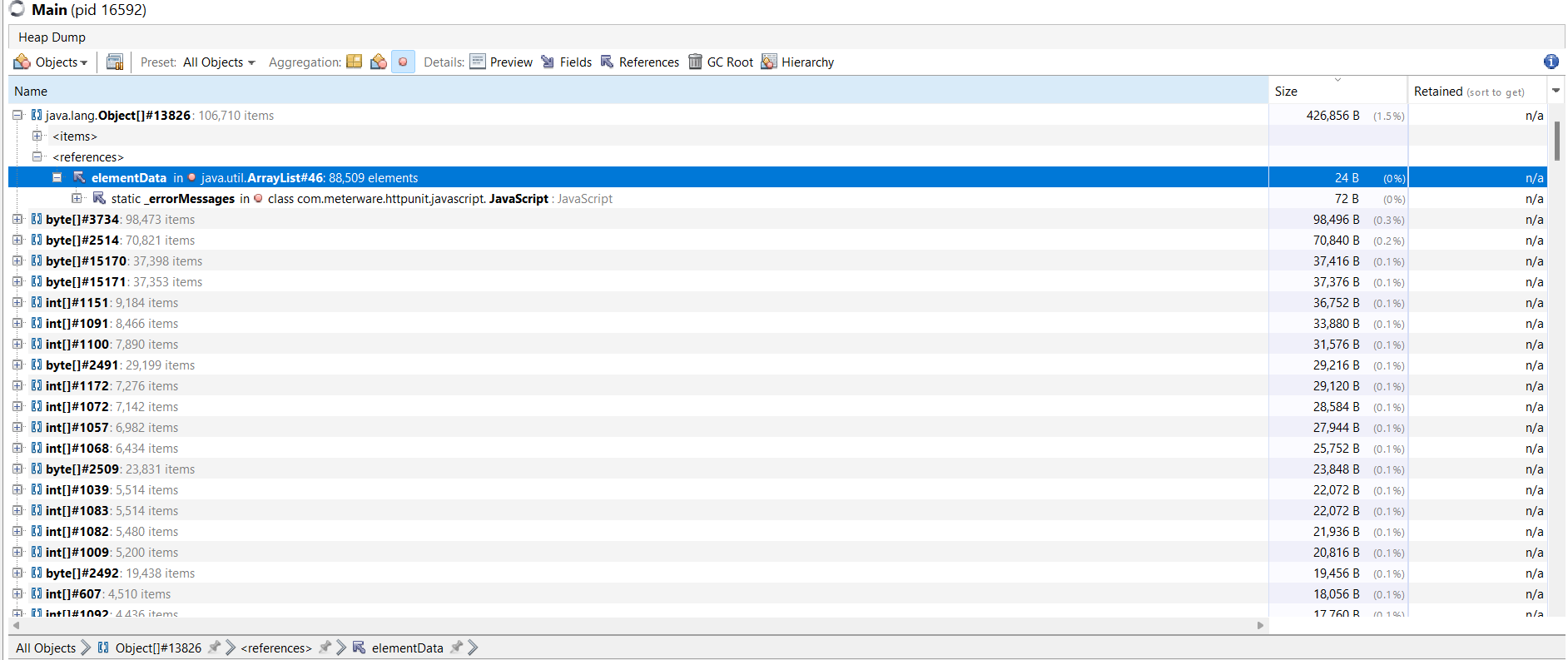


Можно сделать вывод, что утечка памяти действительно существует.

1. Локализация проблемы



Можно увидеть, что больше всего у нас объектов типа byte. Далее за byte, следует String. Чтобы понять, где у нас столько строк, переключаемся в статистику по кол-ву instance-ов и видим, что у нас есть объект ArrayList под названием \_errorMessages, в котором очень много элементов типа String с информацией об ошибках.





Искомный класс – com.meterware.httpunit.javascript.JavaScript, что в нем находится.



Добавление элементов в данный список происходит только в одном методе :



Таким образом, элементы накапливаются в списке при этом нигде не очищаются, что приводится к утечке памяти в связи с отсутствием очистки

1. Устрение проблемы

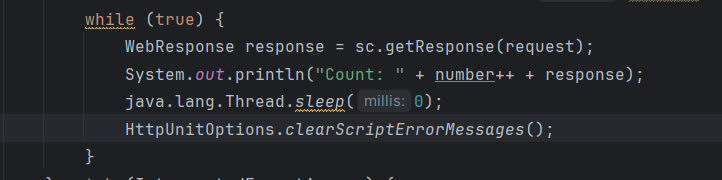
Попробуем найти в этом классе метод для очистки списка:



Данный метод используется в классе com.meterware.httpunit.javascript.JavaScriptEngineFactory:

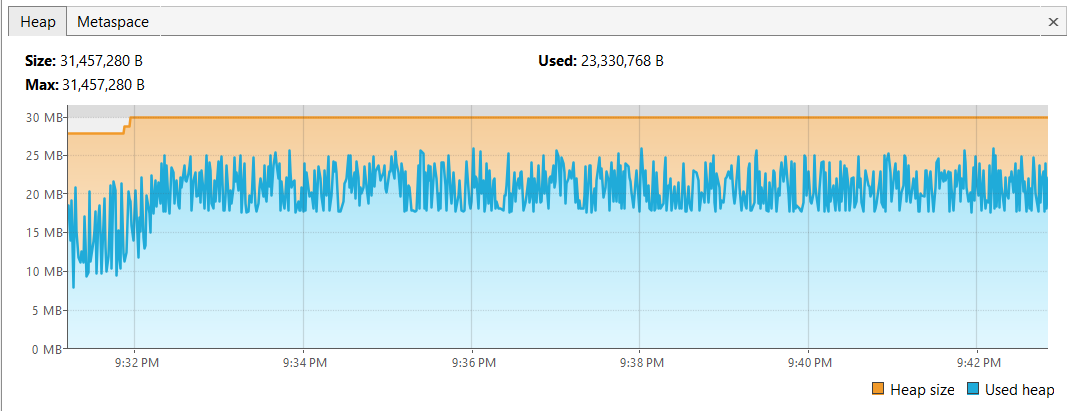


Добавим очистку списка после выполнения каждого запроса в главном цикле программы:



1. Проверка устранения утечки памяти

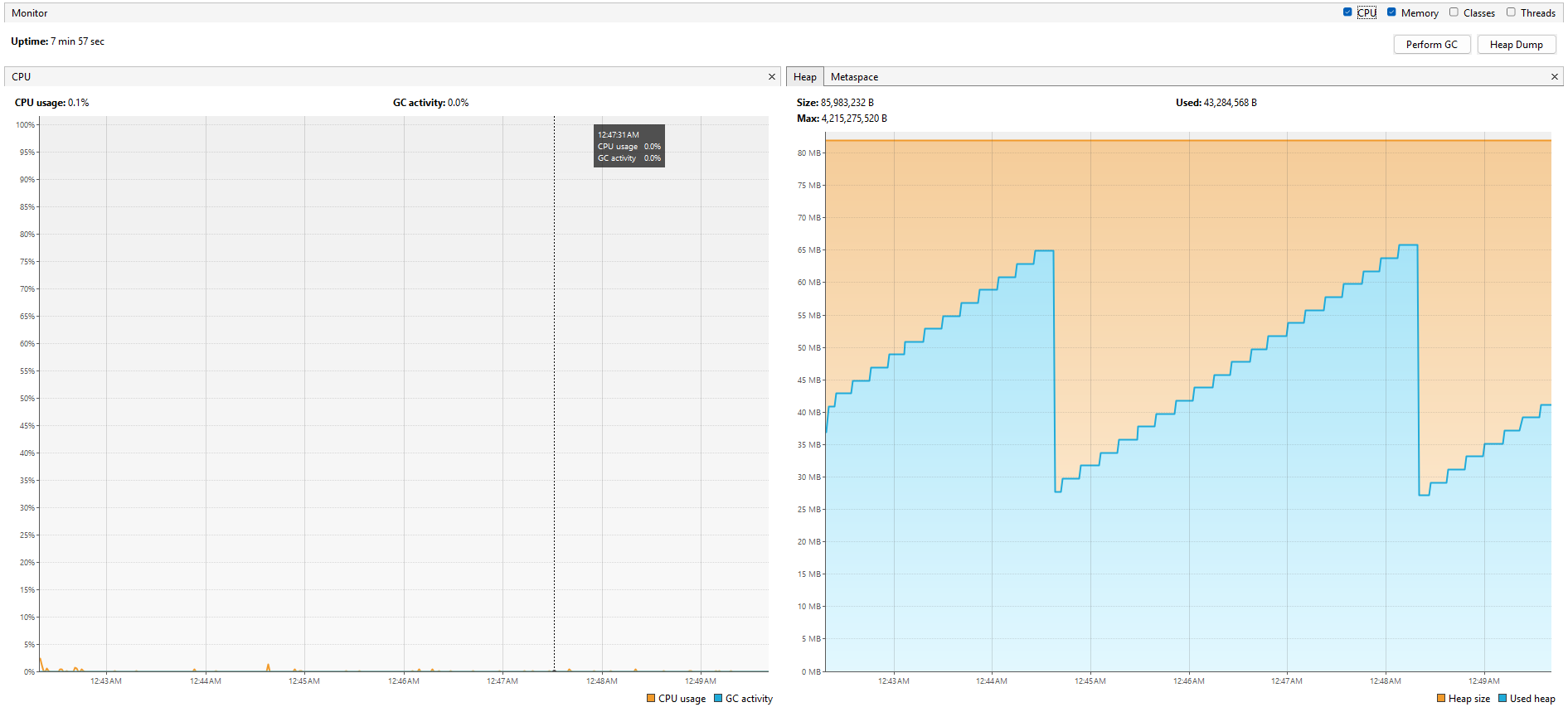
После запуска программы наблюдаем, что теперь не расходуется больше 30 Мб кучи, размер кучи постоянно не растет и сборщик мусора работает в нормальном режиме:



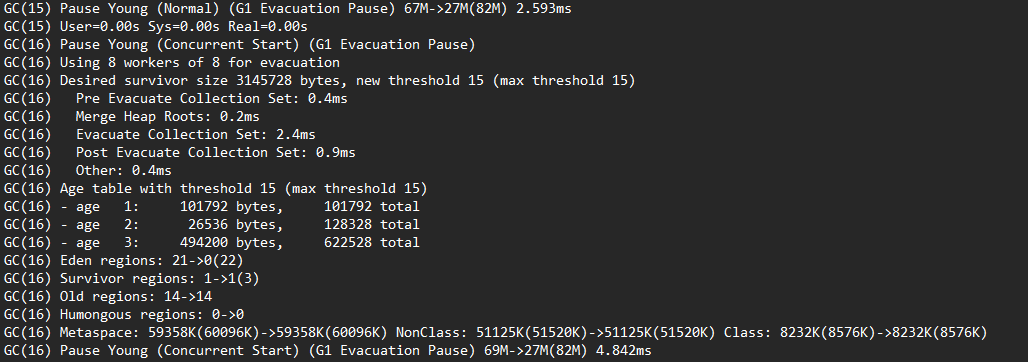
Утечка памяти была успешно устранена.

1. Обнаружение утечек памяти в веб-проектак

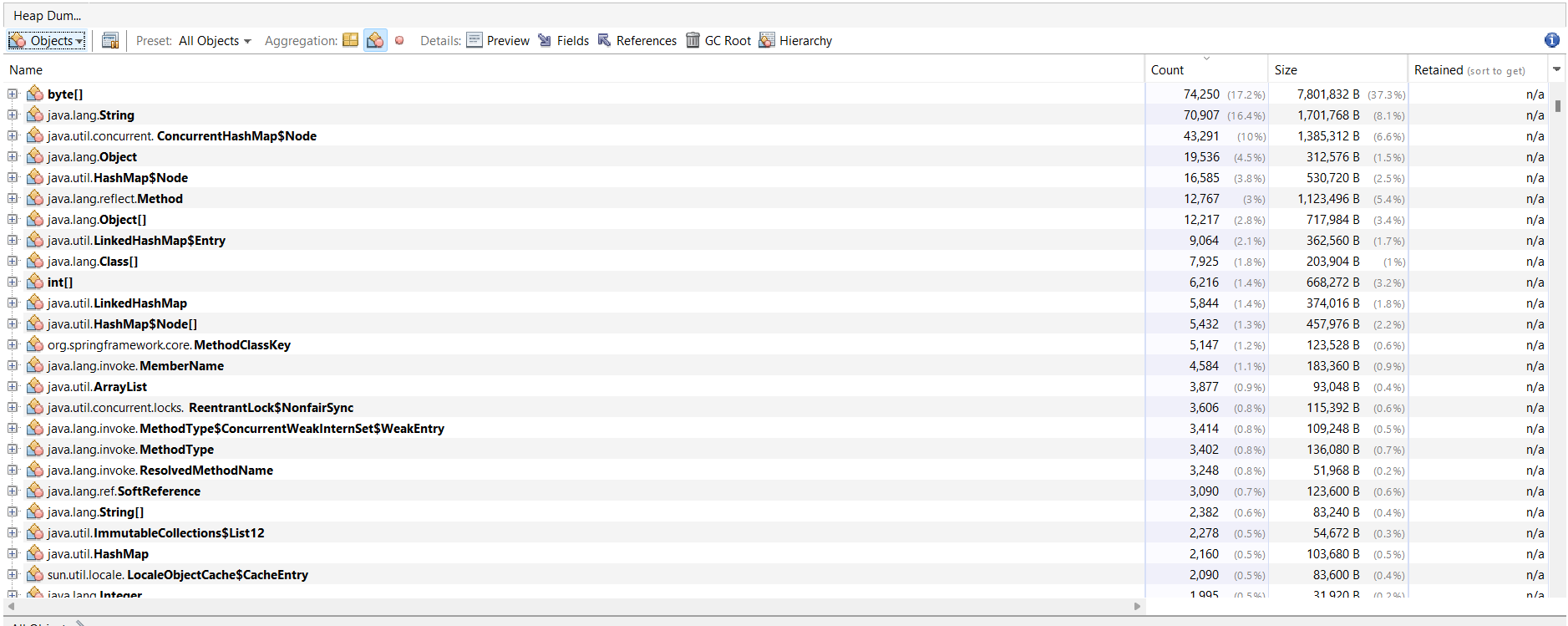
Запускаем проекты и контролируем их в VisualVM

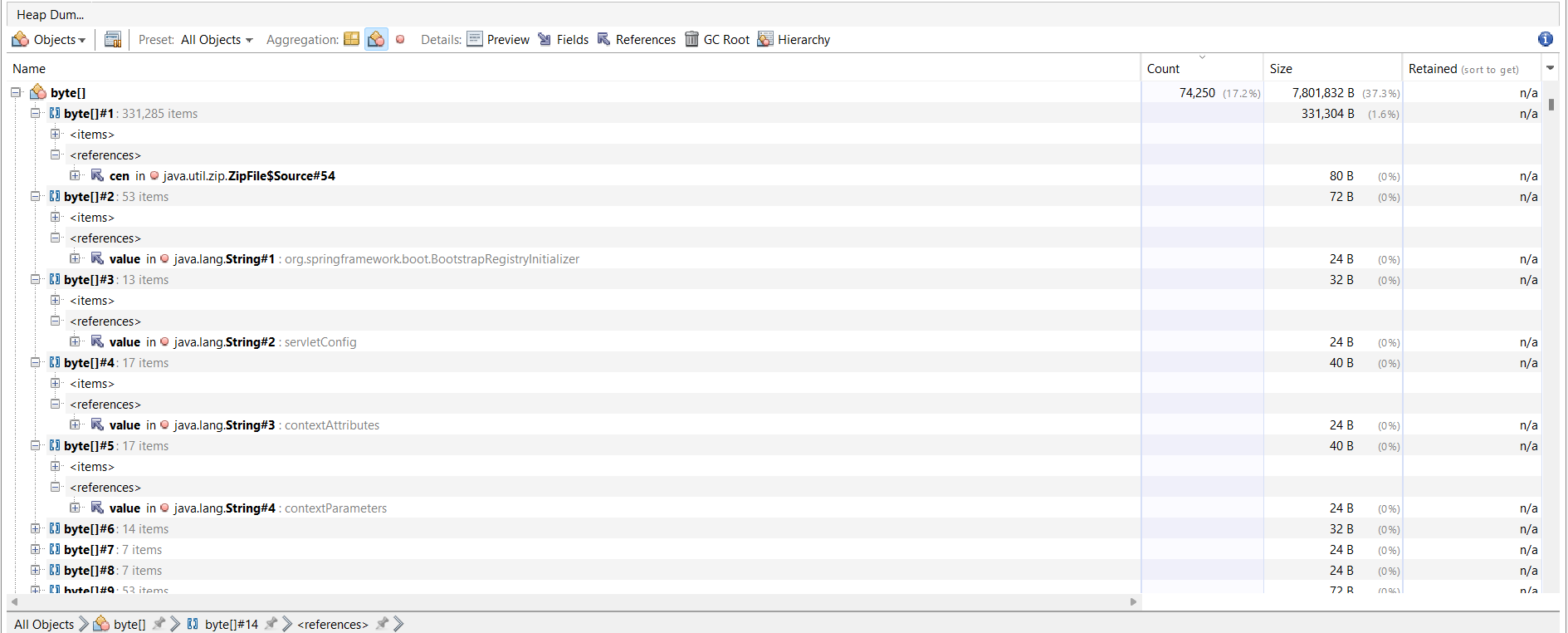


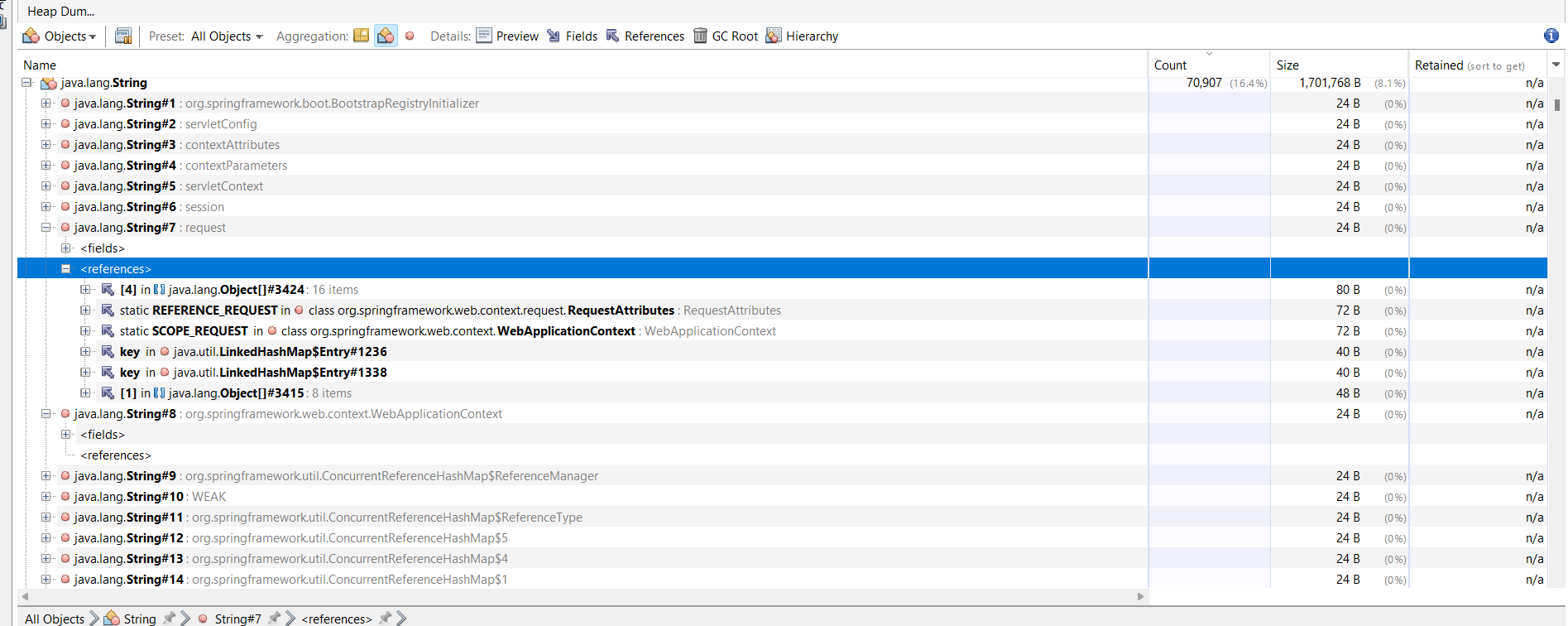
GC работает хорошо, сбор мусора организуется автоматически, поэтому память Heap не будет переполняться.



Посмотрим на распределение памяти в куче, которое использовалось и используется программой.







Утечки памяти нет.

## 6. Вывод

Во время выполнения лабораторной работы мы изучили утилиты JConsole и VisualVM для мониторинга и профилирования Java-приложений. Это помогло нам разобраться в деталях работы JVM и научиться определять, какие компоненты приложения влияют на его производительность. Благодаря этому, мы смогли успешно локализовать и устранить проблемы, связанные с производительностью, на основе собранных данных и анализа.