

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 «Методы и средства программной инженерии»

Выполнили бедные студенты:

Канукова Ева Алановна

Гурьянов Кирилл Алексеевич

Поток: 1.6

Преподаватель:

Райла Мартин

Санкт-Петербург

2023

1. Текст задания

Вариант **52198**

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 количество установленных пользователем точек стало кратно 10, 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 значение переменной classpath для данной 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 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.

2. Исходный код разработанных MBean-классов и сопутствующих классов

```
public class PointCounter extends NotificationBroadcasterSupport implements PointCounterMXBean {
    1 usage
    private int sequenceNumber = 0;
    5 usages
    private int count = 0;
    3 usages
    private int unHitCount = 0;
    1 usage
    private AreaChecker areaChecker = new AreaChecker();
    1 usage
    private static final PointCounter pointCounter = new PointCounter();
    2 usages ▲ Korako
    public static PointCounter getInstance() { return pointCounter; }

    2 usages ▲ Korako *
    @Override
    public void check(double x, double y, double r) {
        count++;
        if (areaChecker.checkHit(x, y, r)) unHitCount++;
        if (count % 10 == 0)
            sendNotification(new Notification(type: "10", source: this, sequenceNumber++, System.currentTimeMillis(), message: "The number of points is " + count));
    }
}
```

```
9     public class ClickIntervalCounter extends NotificationBroadcasterSupport implements ClickIntervalCounterMXB
10
11     private final List<Long> clickTimestamps = new ArrayList<>();
12     private final AtomicLong totalInterval = new AtomicLong( initialValue: 0 );
13     private final AtomicLong clickCount = new AtomicLong( initialValue: 0 );
14
15     private static final ClickIntervalCounter CLICK_INTERVAL_COUNTER = new ClickIntervalCounter();
16     public static ClickIntervalCounter getInstance() { return CLICK_INTERVAL_COUNTER; }
17
18
19     @Override
20     public synchronized void addClick() {
21         long currentTimestamp = System.currentTimeMillis();
22         clickTimestamps.add(currentTimestamp);
23         clickCount.incrementAndGet();
24
25         if (clickTimestamps.size() > 1) {
26             long previousTimestamp = clickTimestamps.get(clickTimestamps.size() - 2);
27             long interval = currentTimestamp - previousTimestamp;
28             totalInterval.addAndGet(interval);
29         }
30     }
31 }
```

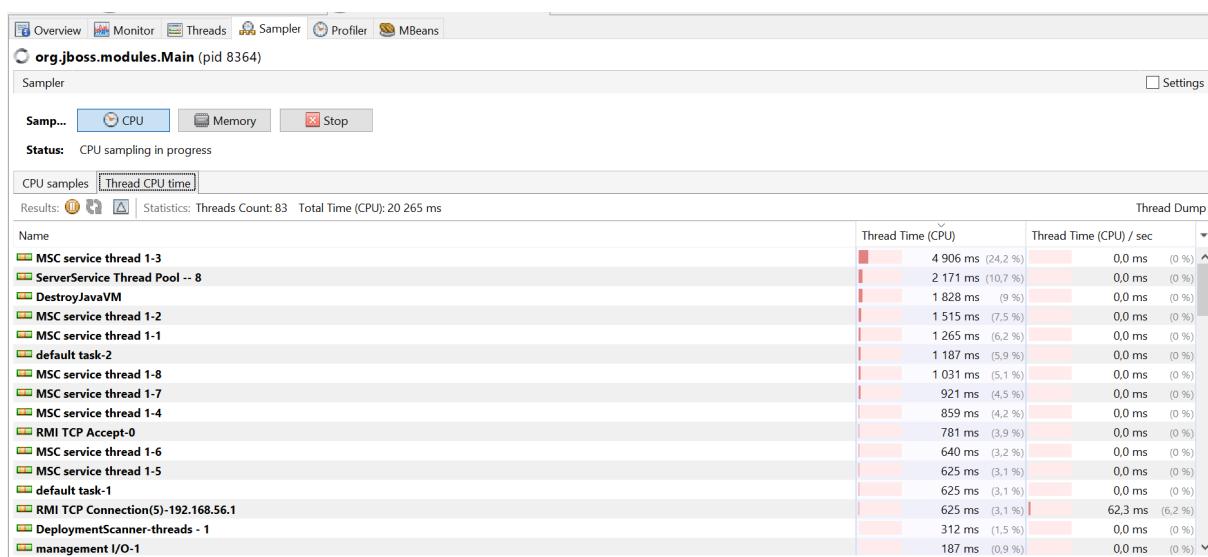
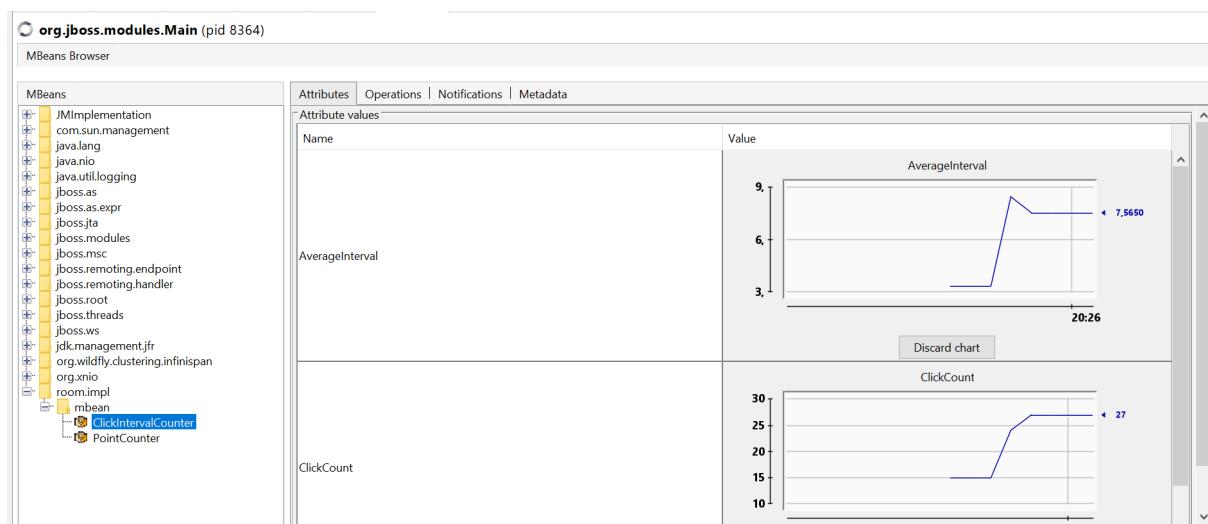
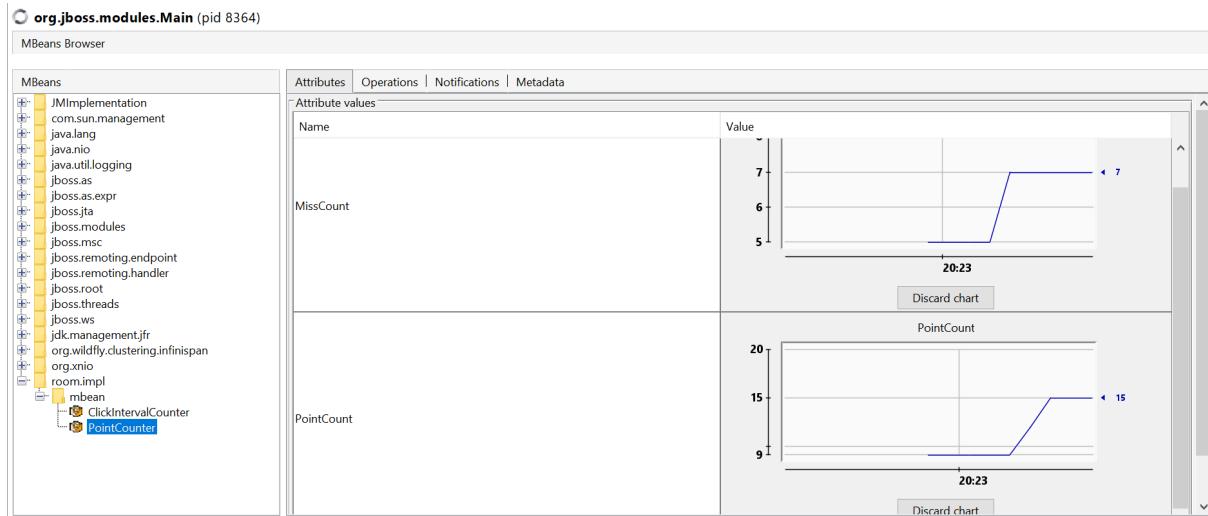
```
38     @Override
39     public double getAverageInterval() {
40         long count = clickCount.get();
41         if (count > 0) {
42             long total = totalInterval.get();
43             return (double) total / count / 1000;
44         }
45         return 0.0;
46     }

```

3. Скриншоты программы JConsole со снятыми показаниями, выводы по результатам мониторинга.

Overview		Memory		Threads		Classes		VM Summary		MBeans																	
<ul style="list-style-type: none"> └ JImplementation └ com.sun.management └ java.lang <ul style="list-style-type: none"> └ ClassLoading └ Compilation └ GarbageCollector └ Memory <ul style="list-style-type: none"> └ MemoryManager └ MemoryPool └ OperatingSystem <ul style="list-style-type: none"> └ Runtime <ul style="list-style-type: none"> └ Attributes <ul style="list-style-type: none"> └ Name <ul style="list-style-type: none"> └ ClassPath └ SpecVersion └ SpecVendor └ StartTime └ StartUpPathSupported └ VmName └ VmVendor └ VmVersion └ LibraryPath └ BootClassPath └ Uptime └ ManagementSpecVersion └ SpecName └ InputArguments └ SystemProperties └ Pid └ ObjectName └ Threading └ java.nio └ java.util.logging └ jboss.as <ul style="list-style-type: none"> └ 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 └ org.xnio └ room.impl <ul style="list-style-type: none"> └ mbean <ul style="list-style-type: none"> └ ClickIntervalCounter <ul style="list-style-type: none"> └ Attributes └ Operations └ Notifications └ PointCounter <ul style="list-style-type: none"> └ Attributes └ Operations └ Notifications 																											
<table border="1"> <thead> <tr> <th colspan="2">Attribute values</th> </tr> <tr> <th>Name</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>Count</td> <td>10</td> </tr> <tr> <td>UnHitCount</td> <td>7</td> </tr> </tbody> </table>												Attribute values		Name	Value	Count	10	UnHitCount	7								
Attribute values																											
Name	Value																										
Count	10																										
UnHitCount	7																										
<table border="1"> <thead> <tr> <th colspan="2">Attribute values</th> </tr> <tr> <th>Name</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>AverageInterval</td> <td>1.2663</td> </tr> <tr> <td>ClickCount</td> <td>10</td> </tr> </tbody> </table>												Attribute values		Name	Value	AverageInterval	1.2663	ClickCount	10								
Attribute values																											
Name	Value																										
AverageInterval	1.2663																										
ClickCount	10																										
<table border="1"> <thead> <tr> <th colspan="2">Attribute value</th> </tr> <tr> <th>Name</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>ClassPath</td> <td>/home/korako2/ideaProjects/wildfly-27.0.0.Final/jboss-modules.jar</td> </tr> </tbody> </table>												Attribute value		Name	Value	ClassPath	/home/korako2/ideaProjects/wildfly-27.0.0.Final/jboss-modules.jar										
Attribute value																											
Name	Value																										
ClassPath	/home/korako2/ideaProjects/wildfly-27.0.0.Final/jboss-modules.jar																										
<table border="1"> <thead> <tr> <th colspan="2">MBeanAttributeInfo</th> </tr> <tr> <th>Name</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>Attribute:</td> <td></td> </tr> <tr> <td>Name</td> <td>ClassPath</td> </tr> <tr> <td>Description</td> <td>ClassPath</td> </tr> <tr> <td>Readable</td> <td>true</td> </tr> <tr> <td>Writable</td> <td>false</td> </tr> <tr> <td>Type</td> <td>java.lang.String</td> </tr> </tbody> </table>												MBeanAttributeInfo		Name	Value	Attribute:		Name	ClassPath	Description	ClassPath	Readable	true	Writable	false	Type	java.lang.String
MBeanAttributeInfo																											
Name	Value																										
Attribute:																											
Name	ClassPath																										
Description	ClassPath																										
Readable	true																										
Writable	false																										
Type	java.lang.String																										
<table border="1"> <thead> <tr> <th colspan="2">Descriptor</th> </tr> <tr> <th>Name</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>Attribute:</td> <td></td> </tr> <tr> <td>openType</td> <td>javax.management.openmbean.SimpleType{name=java.lang.String}</td> </tr> <tr> <td>originalType</td> <td>java.lang.String</td> </tr> </tbody> </table>												Descriptor		Name	Value	Attribute:		openType	javax.management.openmbean.SimpleType{name=java.lang.String}	originalType	java.lang.String						
Descriptor																											
Name	Value																										
Attribute:																											
openType	javax.management.openmbean.SimpleType{name=java.lang.String}																										
originalType	java.lang.String																										

4. Скриншоты программы VisualVM со снятыми показаниями, выводы по результатам профилирования.



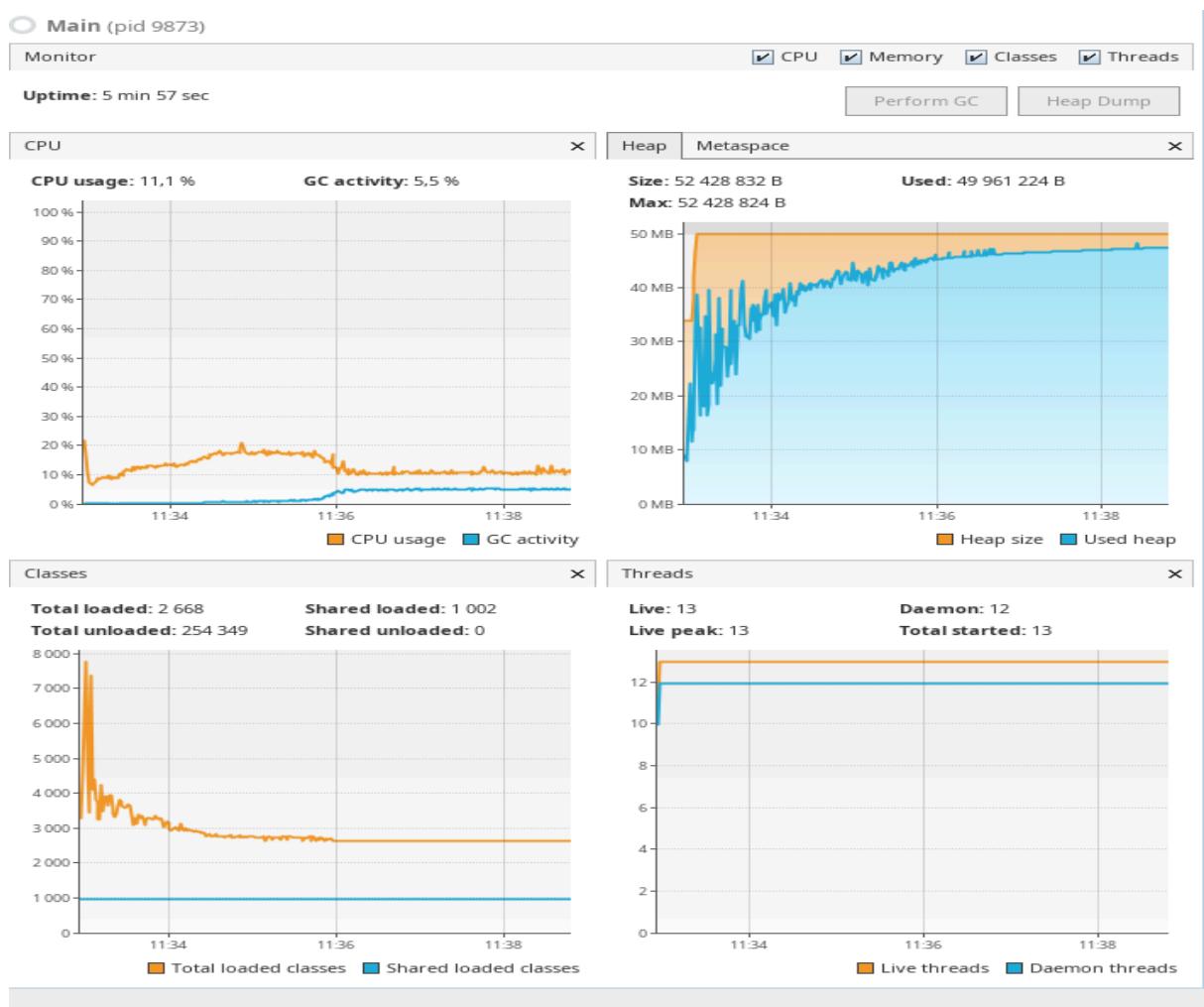
Наибольший процент времени CPU занимает поток MSC

service thread 1-3.

Этот поток является частью механизма управления сервисами, используемого контейнером приложений для запуска, остановки и перезапуска компонентов приложения, таких как EJB-бины или сервлеты.

5. Скриншоты программы VisualVM с комментариями по ходу поиска утечки памяти.

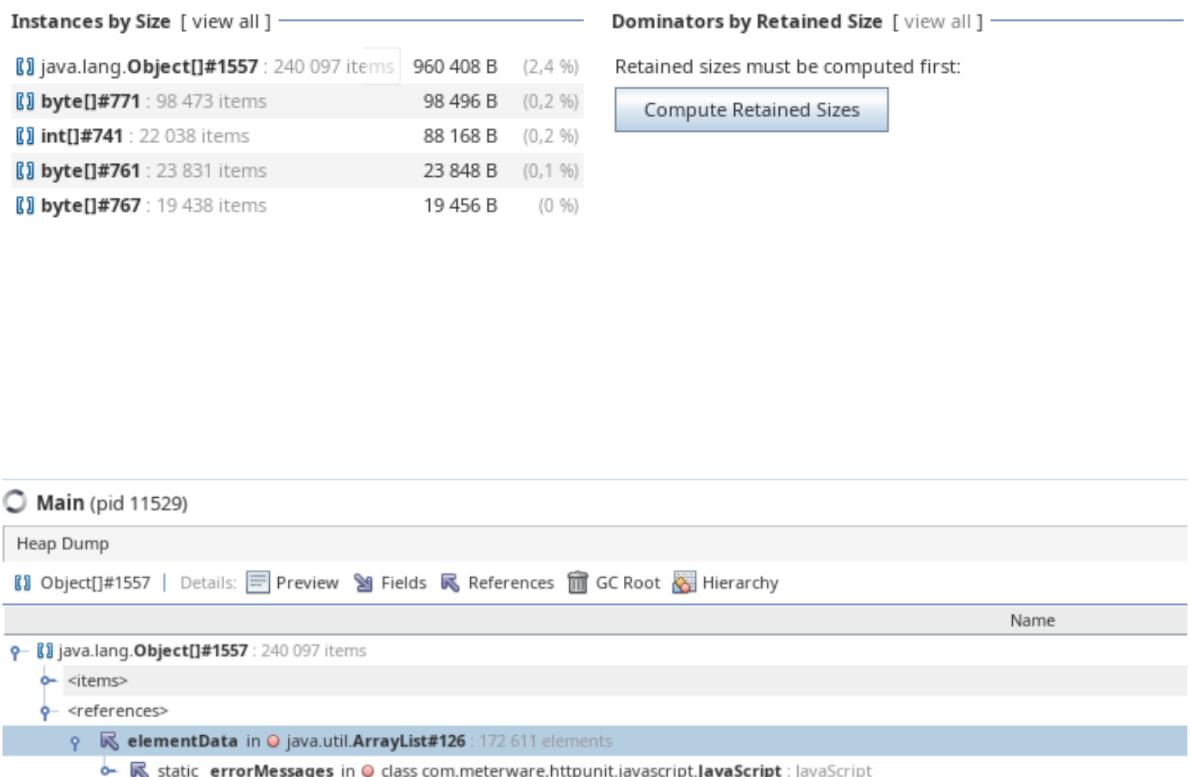
Спустя 5 минут 57 секунд после запуска программы, было выброшено исключение: java.lang.OutOfMemoryError



Из графика кучи в VisualVM мы можем видеть, что используемый размер кучи постоянно рос и по итогу куча

переполнилась. Следовательно можно сделать вывод, что мы действительно имеем утечку памяти.

Попробуем найти корень проблемы. Для этого проанализируем Heap dump. Можно заметить, что `java.lang.Object[]#1557` занимает наибольшее количество памяти:



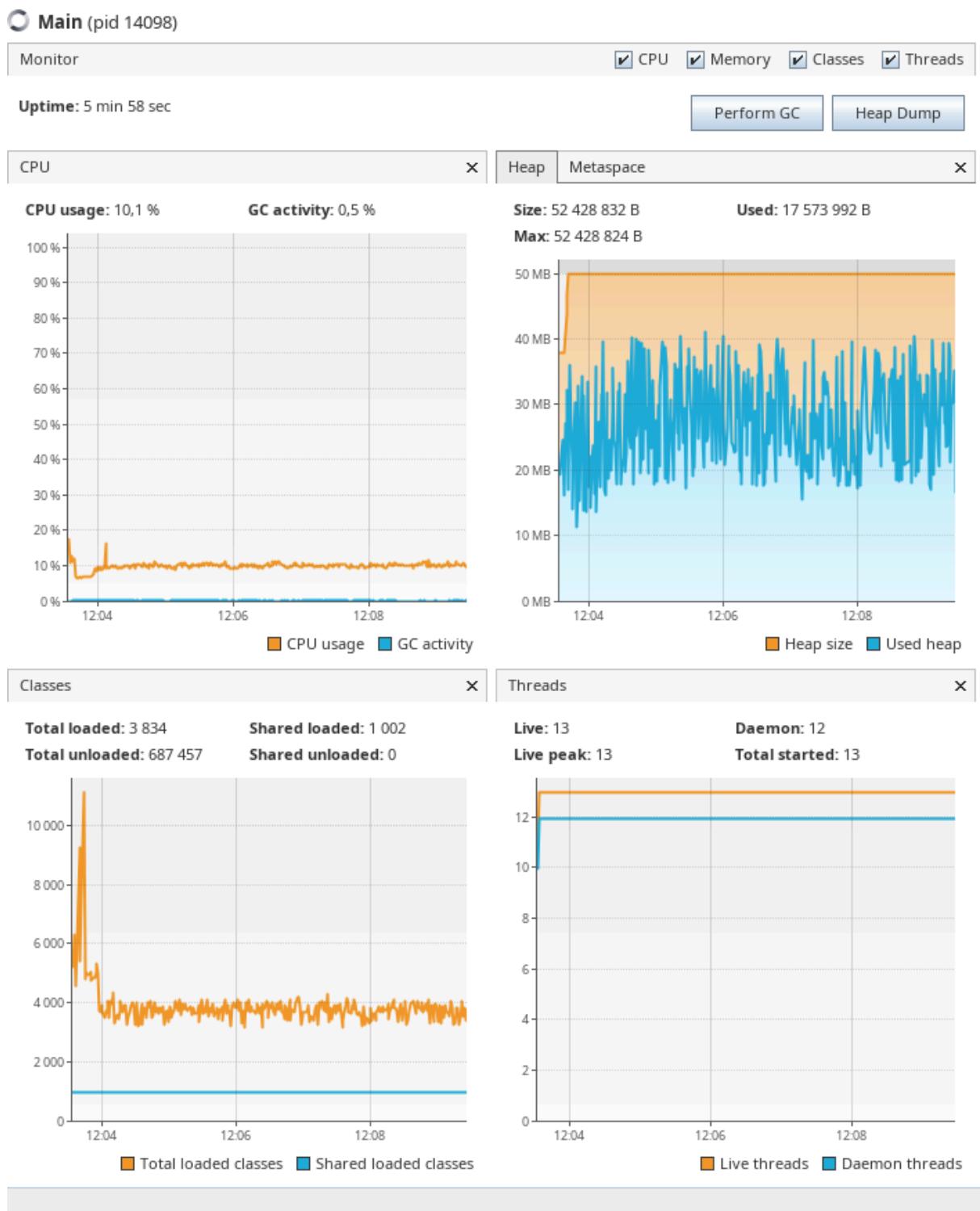
Также мы видим, что эти объекты находятся внутри `ArrayList`, который называется `_errorMessages`, находящийся в классе: `com.meterware.httpunit.javascript.JavaScript`.

```
3 usages
195     ↘
196         ↘
197             ↘
198                 ↘
199                     ↘
200                         ↘
201                             ↘
202                                 ↘
203                                     ↘
204                                         ↘
205                                             ↘
206                                                 ↘
207             ↗
    }
```

Мы можем заметить, что подавление элементов в `_errorMessages` происходит только в этом месте. Элементы только добавляются в `ArrayList`, однако не очищаются, что приводит к утечке памяти. Исправим это путем вызова метода `clearScriptErrorMessages()` в классе `HttpUnitOptions`.

```
int count = 0;
while (true) {
    WebResponse response = sc.getResponse(request);
    System.out.println("Count: " + number++ + response);
    java.lang.Thread.sleep( millis: 0 );
    count++;
    if (count == 100) {
        HttpUnitOptions.clearScriptErrorMessages();
        count = 0;
    }
}
```

После этого можем видеть, что куча перестала переполняться, утечка устранена.



6. Выводы по работе.

В ходе данной лабораторной работы мы ознакомились с утилитами JConsole и VisualVM для мониторинга и профилирования

Java приложений. Также мы научились локализовывать и устранять проблемы, связанные с производительностью приложения.

