

**Университет ИТМО**

**Факультет программной инженерии и компьютерных технологий**

**Лабораторная работа №4 по Методам и Средствам Программной Инженерии**

**Выполнил: Богатов Александр Сергеевич**

**Группа: Р3233**

**Вариант: 6699**

**Преподаватель: Цопа Евгений Алексеевич**

**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 15, разработанный MBean должен отправлять оповещение об этом событии.
- MBean, определяющий процентное отношение "промахов" к общему числу кликов пользователя по координатной плоскости.

2. С помощью утилиты JConsole провести мониторинг программы:

- Снять показания MBean-классов, разработанных в ходе выполнения задания 1.
- Определить количество классов, загруженных в JVM в процессе выполнения программы.

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.

### MBean методы:

```
@Override
public long getResultAmount () {
    loadResults ();
    return results.size();
}
```

```

@Override
public long getSVGResultAmount() {
    loadResults();
    return results.stream().filter(result -> result.getType().equals("fromSVG")).count();
}

@Override
public long getMissAmount() {
    loadResults();
    return results.stream().filter(result -> result.getHit() == false).count();
}

@Override
public long checkPointAmountDivisor() {
    long amount = getResultAmount();
    long misses = getMissAmount();
    resultAmount = amount;
    missAmount = misses;
    if (amount % 15 == 0) {
        sendNotification(new Notification("Result amount can be divided by 15", this.getClass().getName(),
sequenceNumber++, "Overall number of results: " + results.size() + "\n Missed results: " + misses));
    }
    return amount;
}

@Override
public double getMissPercentage() {
    long amount = getSVGResultAmount();
    long misses = getMissAmount();
    svgAmount = amount;
    missAmount = misses;
    if (amount != 0) {
        missPercentage = misses * 100 / amount;
        return misses * 100 / amount;
    }
    missPercentage = 0;
}

```

```
        return 0;
    }
}
```

### MXBeanContextListener:

```
package management;

import data.ResultBean;

import javax.management.*;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import java.lang.management.ManagementFactory;

public class MXBeanContextListener implements ServletContextListener {

    @Override
    public void contextInitialized(ServletContextEvent sce) {
        try {
            MBeanServer = ManagementFactory.getPlatformMBeanServer();
            ResultBean bean = new ResultBean();
            ObjectName beanObjName = new ObjectName("data:type=mbeans,name=result");
            mBeanServer.registerMBean(bean, beanObjName);
            MXBeanListener beanListener = new MXBeanListener();
            mBeanServer.addNotificationListener(beanObjName, beanListener,
            beanListener.getNotificationFilter(), null);
        } catch (InstanceAlreadyExistsException | MalformedObjectNameException | MBeanRegistrationException |
            NotCompliantMBeanException e) {
            e.printStackTrace();
        } catch (InstanceNotFoundException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void contextDestroyed(ServletContextEvent sce) {
    }
}
```

```

    try {
        ManagementFactory.getPlatformMBeanServer().unregisterMBean(new
ObjectName("data:type=mbeans,name=result"));
    } catch (MBeanRegistrationException | InstanceNotFoundException | MalformedObjectNameException e) {
        e.printStackTrace();
    }
}
}

```

### Показания MBeans:

Attribute values	
Name	Value
MissAmount	<b>46</b>
MissPercentage	<b>47.0</b>
ResultAmount	<b>150</b>
SVGResultAmount	<b>96</b>

Notification buffer					
TimeStamp	Type	UserData	SeqNum	Message	Event
23:11:12:209	Result amount can be divided by 15		0	Overall number of results: 135 Missed results: 40	javax.management.Notification[source=data.R... data.ResultBean

### Изменение атрибутов MBean:

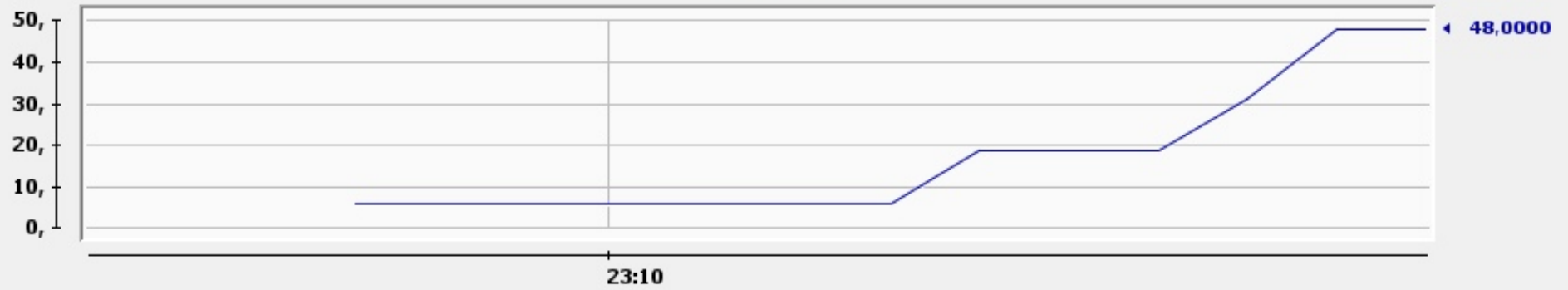
Попользуемся разработанным сайтом какое-то время и посмотрим на то, как изменяются значения атрибутов MBeans.

MissAmount

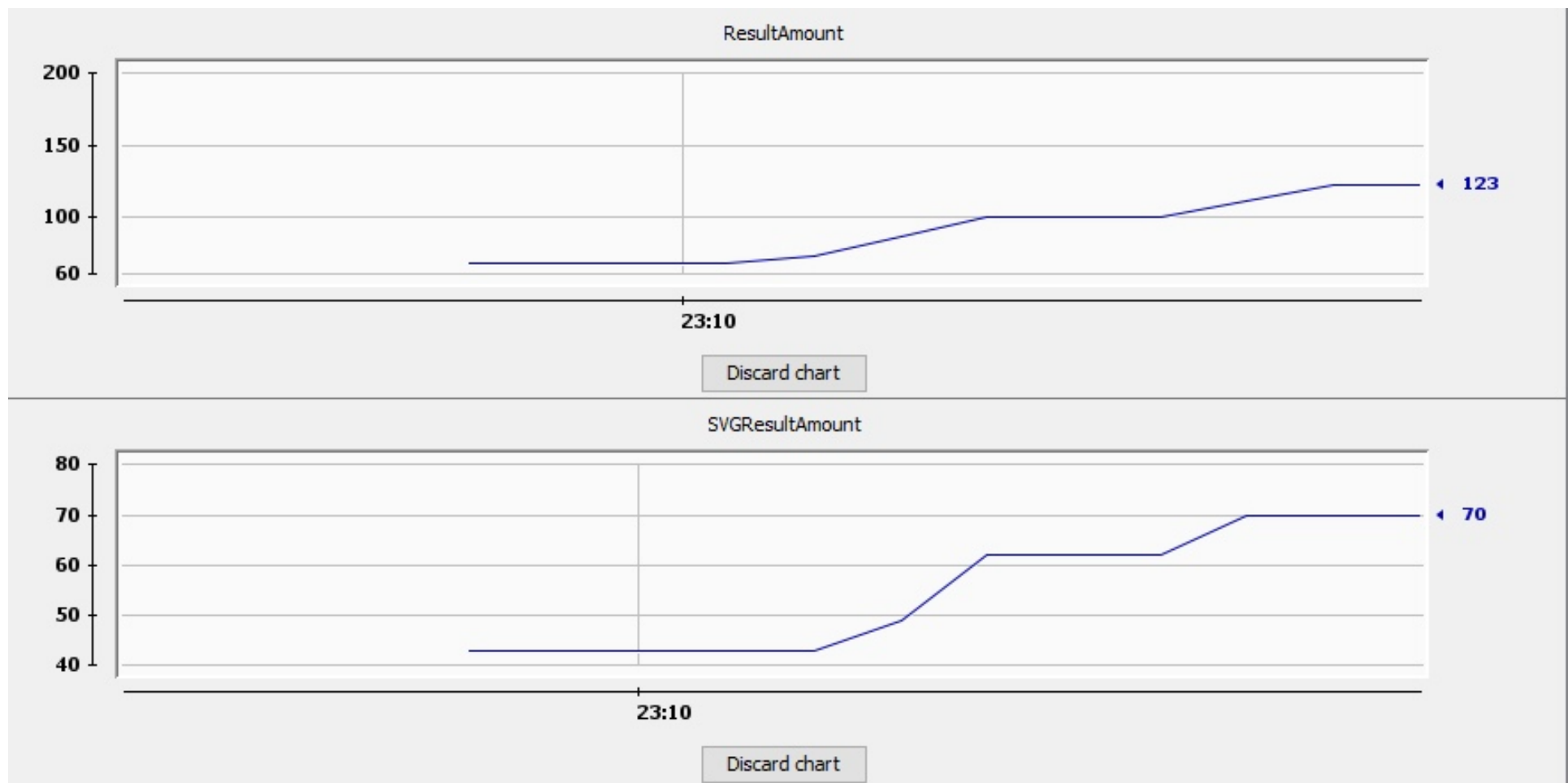


Discard chart

MissPercentage



Discard chart



### Количество классов в JVM:

Информацию о числе загруженных и выгруженных из JVM классов можно найти во вкладке Classes.

Time: 2022-05-25 23:25:42  
Current classes loaded: 34 412  
Total classes loaded: 34 418  
Total classes unloaded: 6

Класс, занимающий больше всего памяти:

Сделав дамп кучи, выясняем, что больше всего места в JVM занимают объекты HashMap\$Node. Посмотрим корни GC и выясним, что наибольшее число объектов используются JBoss'ом.

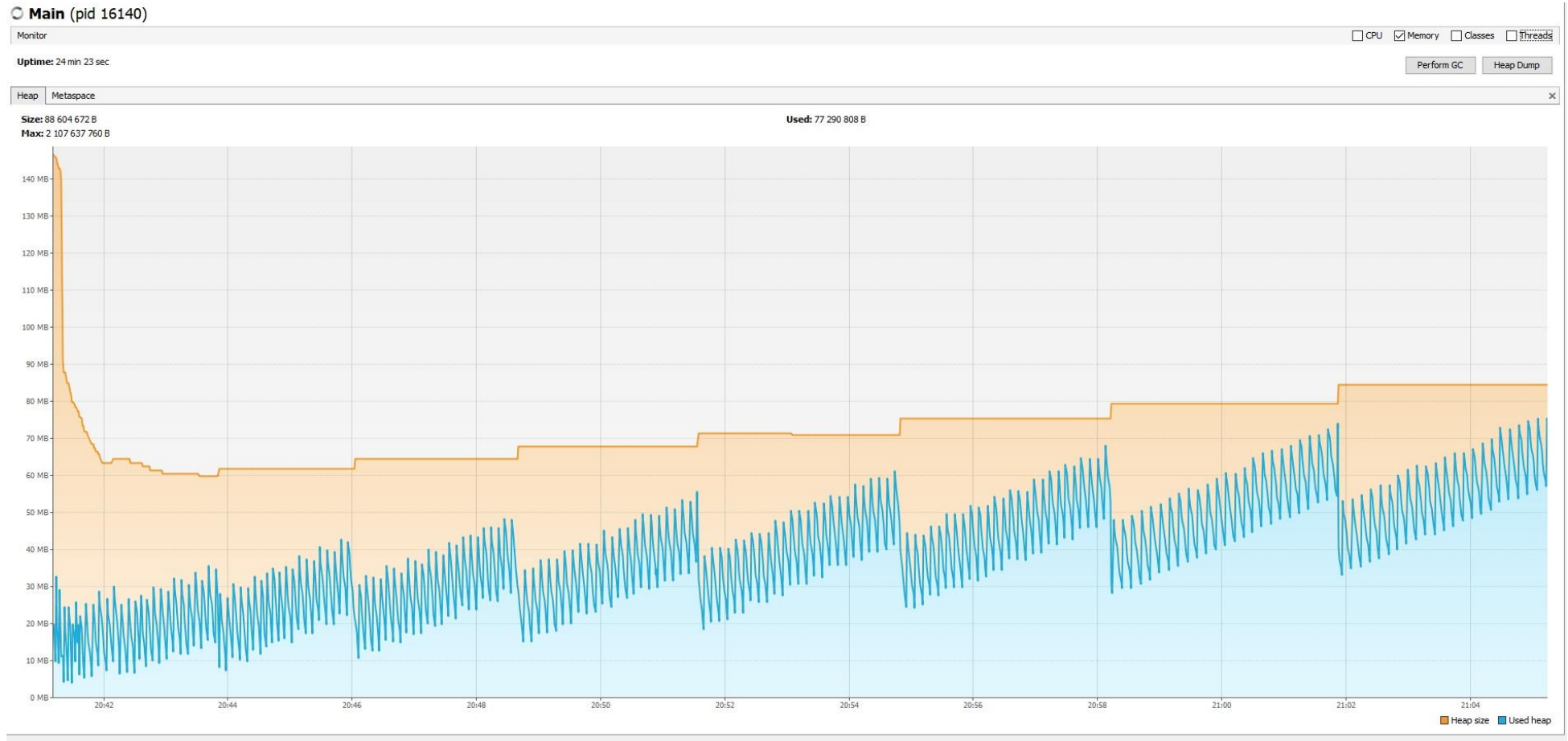
Heap Dump			
Objects ▾   Preset: All Objects ▾   Aggregation:    Details:  Preview  Fields  References  GC Root  Hierarchy			
Name	Count	Size	Retained (sort to get)
java.util.HashMap\$Node	224 300 (14,2%)	9 869 200 B (6,4%)	n/a
char[]	222 343 (14,1%)	25 623 886 B (16,5%)	n/a
java.lang.String	221 033 (14%)	6 188 924 B (4%)	n/a
java.lang.Object[]	51 337 (3,3%)	4 475 192 B (2,9%)	n/a
byte[]	49 403 (3,1%)	57 691 361 B (37,2%)	n/a
java.util.HashMap	37 355 (2,4%)	2 390 720 B (1,5%)	n/a
java.lang.ref.Finalizer	31 382 (2%)	2 008 448 B (1,3%)	n/a

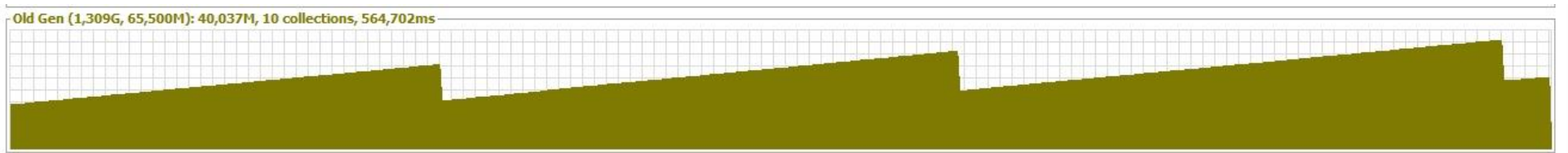
GC Root	
Name	Count
org.jboss.modules.ModuleClassLoader#103 [GC root - Java frame]	114 521 (7,3%)
org.jboss.msc.service.ServiceContainerImpl\$ServiceThread#1 [GC root - Java frame, thread object] : MSC service thread 1-8	30 418 (1,9%)
<no GC root>	24 077 (1,5%)
java.util.concurrent.ForkJoinWorkerThread#1 [GC root - Java frame, thread object] : ForkJoinPool.commonPool-worker-7	9 279 (0,6%)
org.jboss.as.weld.services.bootstrap.WeldExecutorServices\$WeldExecutor#1 [GC root - Java frame]	4 922 (0,3%)
java.lang.Thread#17 [GC root - Java frame, thread object] : RxCachedWorkerPoolEvictor-1	3 183 (0,2%)
org.xnio.nio.WorkerThread#5 [GC root - Java frame, thread object] : XNIO-1 Accept	3 129 (0,2%)
class java.sql.DriverManager [GC root - sticky class] : DriverManager	3 023 (0,2%)
com.sun.jmx.remote.internal.ArrayNotificationBuffer#1 [GC root - Java frame, monitor used]	3 016 (0,2%)
java.util.TimerThread#3 [GC root - Java frame, thread object] : jaeger.RemoteReporter-FlushTimer	2 774 (0,2%)
java.util.TreeSet#481 [GC root - Java frame] : 1 element	2 602 (0,2%)
org.wildfly.common.ref.Reference\$ReaperThread#1 [GC root - Java frame, thread object] : Reference Reaper #3	2 503 (0,2%)



## Исправление утечки:

Проведем наблюдение за кучей программы: занятое место в куче с каждой мажорной очисткой становится больше, что свидетельствует об утечке памяти.





Сделав дамп кучи видим, что много места занимают объекты `char[]`. Посмотрим на корни GC объектов и обнаружим там пользовательский класс `com.meterware.httpunit.javascript`.

Classes

Class Name	Instances [%]	Instances	Size
<b>char[]</b>		56 375 (21,8%)	10 917 138 (53,8%)
java.lang.String		56 109 (21,7%)	1 571 052 (7,7%)
java.util.TreeMap\$Entry		35 089 (13,6%)	2 000 073 (9,9%)
java.lang.Long		22 221 (8,6%)	533 304 (2,6%)
java.util.TreeMap		10 978 (4,3%)	878 240 (4,3%)

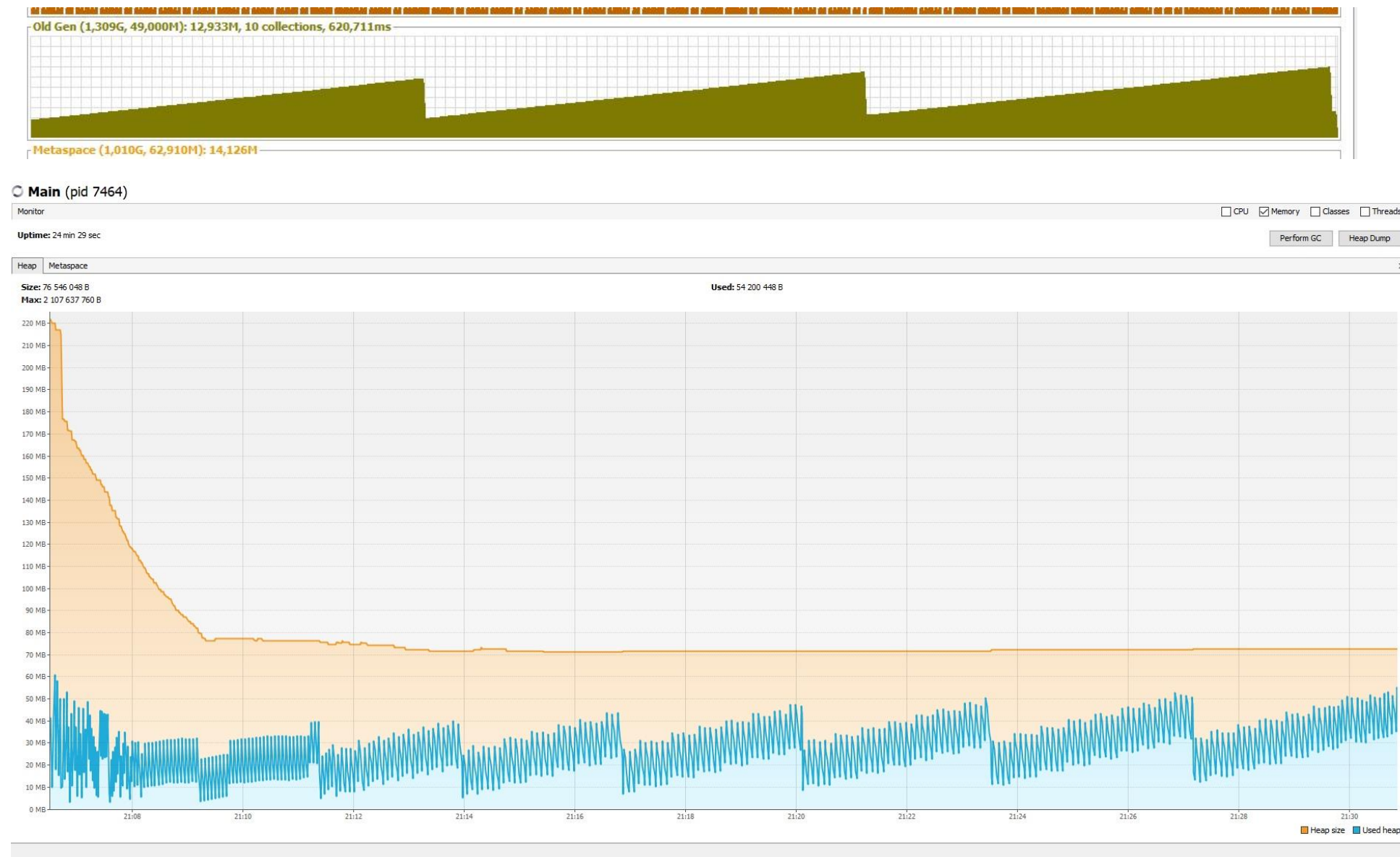
References

Field	Type	Value
this	char[]	#4625 Script 'document.wr('Hello Do...
value	String	#4750 Script 'document.wr('Hello Do...
[123]	Object[]	#5350 47 427 items
elementData	ArrayList	#28
_errorMessages	JavaScript	class JavaScript
[54]	Object[]	#361 640 items
elementData	Vector	#24
classes (Java frame)	Launcher\$AppClassLoader	#1

Среди полей класса есть Array List хранящий ошибки исполнения JS, но никогда не очищающийся. Также содержимое поля никогда не используется в программе. Изменим метод так, чтобы в списке максимально хранилось 50 последних ошибок.

```
} else {  
    if (_errorMessages.size() >= 50) {  
        _errorMessages.remove( index: 0);  
    }  
    _errorMessages.add( errorMessage );  
}
```

Наблюдаем за исполнением и видим улучшения.



**Вывод:**

При выполнении данной лабораторной работы были изучены методы мониторинга и профилирования Java приложений с помощью утилит JConsole и VisualVM.